

Traffic De-Anonymizer

A. Nur Zincir-Heywood
Vahid Aghaei

Prepared By:
Faculty of Computer Science
Dalhousie University
6050 University Avenue
Halifax, NS B3H 1W5

Contract Reference Number: CSSP-2013-CD-1085

CSA: Rodney Howes, DRDC – Centre for Security Science, 613-943-2474

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

Defence Research and Development Canada

Contract Report
DRDC-RDDC-2014-C277
November 2014

IMPORTANT INFORMATIVE STATEMENTS

CSSP-2013-CD-1085 Traffic De-Anonymizer was supported by the Canadian Safety and Security Program which is led by Defence Research and Development Canada's Centre for Security Science in partnership with Public Safety Canada. The project was led by Public Safety Canada in partnership with Dalhousie University.

Canadian Safety and Security Program is a federally-funded program to strengthen Canada's ability to anticipate, prevent/mitigate, prepare for, respond to, and recover from natural disasters, serious accidents, crime and terrorism through the convergence of science and technology with policy, operations and intelligence.

© Her Majesty the Queen in Right of Canada, as represented by the Minister of National Defence, 2014

© Sa Majesté la Reine (en droit du Canada), telle que représentée par le ministre de la Défense nationale, 2014

Traffic De-Anonymizer

Prepared by:

A. Nur Zincir-Heywood

Vahid Aghaei

Faculty of Computer Science

Dalhousie University 6050 University Avenue

Halifax, NS B3H 1W5

Scientific Authority:

Rodney Howes

DRDC Centre for Security Science

613-943-2474

The scientific or technical validity of this Contract Report is entirely the responsibility of the Contractor and the contents do not necessarily have the approval or endorsement of the Department of National Defence of Canada.

Defence Research and Development

Canada DRDC-RDDC-2014-C277

November 2014

IMPORTANT INFORMATIVE STATEMENTS

CSSP-2013-CD-1085 Traffic De-Anonymizer was supported by the Canadian Safety and Security Program which is led by Defence Research and Development Canada's Centre for Security Science in partnership with Public Safety Canada. The project was led by Public Safety Canada in partnership with Dalhousie University.

Canadian Safety and Security Program is a federally-funded program to strengthen Canada's ability to anticipate, prevent/mitigate, prepare for, respond to, and recover from natural disasters, serious accidents, crime and terrorism through the convergence of science and technology with policy, operations and intelligence.

© Her Majesty the Queen in Right of Canada, as represented by the Minister of National Defence, 2014

© Sa Majesté la Reine (en droit du Canada), telle que représentée par le ministre de la Défense nationale, 2014

FINAL PROJECT REPORT:

Traffic De-Anonymizer

Contract Number: 78820-13-0017

Project Leader: A. Nur Zincir-Heywood
zincir@cs.dal.ca

Graduate Students: Vahid Aghaei

Date: March 28th, 2014

Network Information Management and Security Group

Faculty of Computer Science
Dalhousie University
6050 University Avenue

Halifax, NS
B3H 1W5

ABSTRACT

Proxies are used commonly on today's Internet. On one hand, end users can choose to use proxies for keeping their privacy and ubiquitous systems can use it for intercepting the traffic for purposes such as caching. On the other hand, attackers can use such technologies to anonymize their malicious behaviours. Thus, the prevalence of proxies and the different applications and users connected through a proxy has implications in terms of the different behaviours seen on the network. This is important for defense applications since it can facilitate the assessment of security threats. Thus, systems that can identify infected computers behind a proxy based on their behaviour represent a first step in taking the appropriate actions, for example, when a botnet client is identified. The objective of this research includes identifying proxies and the computers behind them based on their behavior from the traffic log files of a computer, which is on the network that is outside of the proxy. This is what we mean by traffic de-anonymizer. To achieve this: (i) we employ a mixture of log files to represent real-life proxy behavior, and (ii) we design and develop a data driven machine learning based approach to provide recommendations for the automatic identification of computers behind an anonymous proxy. Our results show that we are able to achieve our objectives with a promising performance even though the problem is very challenging.

Table of Contents

1. BACKGROUND	9
2. SQUID PROXY SERVER.....	10
3. GENERATING PROXY DATA SETS.....	13
4. STATE OF THE ART METHODS FOR DETECTING PROXY TRAFFIC	24
4.1. Active Measurement Based Schemes	24
4.2. Passive Measurement Based Schemes.....	25
4.2.1. OWD – One Way Delay	26
4.2.2. Single Measuring Point.....	26
4.2.3. SPP – Synthetic Packet Pairs	26
4.2.4. TTL – Time To Live	27
4.2.5. TTL and OS fingerprinting	27
5. Machine Learning Based Approach.....	28
5.1. Decision Tree Algorithm	28
5.2. Naïve Bayes Algorithm.....	30
6. EVALUATIONS	32
7. EXPERIMENTS AND RESULTS	35
7.1. Results of the Classification Experiments.....	36
7.1.1. “NoProxy-Unencrypted” vs “Proxy-Unencrypted”: Binary Classification	38
7.1.2. “NoProxy-Encrypted” vs “Proxy-Encrypted”: Binary Classification.....	39
7.1.3. “NoProxy” vs “Proxy”: Binary Classification	40
7.1.4. “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted”: 3 Classes Classification.....	41
7.1.5. “Proxy-Encrypted” vs “Proxy-Unencrypted”: Binary Classification	42
7.1.6. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote”: 3 Classes Classification	43
7.1.7. “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote”: 3 Classes Classification.....	44
7.1.8. “NoProxy” vs “Proxy-Local” vs “Proxy-Remote”: 3 Classes Classification	46
7.1.9. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache”: 3 Classes Classification	47
7.1.10. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader”: 3 Classes Classification	48

7.1.11. “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader”: Binary Classification	50
7.2. Analysis of results	51
8. PROPOSED SYSTEM	53
8.1. First Step – Data Preparation:	55
8.2. Second Step – Proxy Classification:	55
8.3. Third Step – Analyzing Proxy Behavior:	56
9. CONCLUSIONS AND FUTURE WORKS	60
References:.....	62

List of Figures

Figure 1: How Squid Proxy Server Works [2].....	12
Figure 2: Our Testbed Network for Generating the Proxy traffic.....	13
Figure 3: The Categories of the Generated Unencrypted Proxy Datasets in the NIMS Lab.	15
Figure 4: A Sample of the http User Agent without Using Proxy	16
Figure 5: A Sample of the http User Agent in a Proxy Network with Default Configuration.....	16
Figure 6: Visiting whatismyip Website (URL-2) on a Direct Network.....	17
Figure 7: Visiting whatismyip Website (URL-2) on a Proxy Network with Default Configuration.....	18
Figure 8: Viewing at http User Agent on Wireshark	19
Figure 9: Hiding the Proxy Server information in the http User Agent.....	21
Figure 10: Visiting whatismyip Website When the Proxy Server in Configured to Hide Itself.....	22
Figure 11: Construction of a classification tree [12].....	30
Figure 12: An Overview of Our Prototype System.....	33
Figure 13: A screenshot of the Prototype GUI	54
Figure 14: A screenshot of the window for selecting the views	56
Figure 15: Rectangle view for the Proxy classification of the NIMS Proxy Dataset.....	57
Figure 16: Tree view for classification of the NIMS Proxy Datasets	58
Figure 17: A Screenshot of the Window of selecting the Log File Category	58
Figure 18: A sample of Proxy Traffic Log File	59

List of Tables

Table 1: Summary of the Generated Unencrypted Proxy Datasets in the NIMS Lab	22
Table 2: Summary of the Generated Encrypted Proxy Datasets in the NIMS Lab.....	23
Table 3: Features Employed in This Research.....	34
Table 4: The Number of Instances (flows) in our Datasets	37
Table 5: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted” Classification.....	38
Table 6: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted” Classification	38
Table 7: Dataset Numbers for “NoProxy-Encrypted” vs “Proxy-Encrypted” Classification	39
Table 8: Results of “NoProxy-Encrypted” vs “Proxy-Encrypted” Classification	39
Table 9: Dataset Numbers for “NoProxy” vs “Proxy” Classification.....	40
Table 10: Results of “NoProxy” vs “Proxy” Classification.....	40
Table 11: Dataset Numbers for “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification	41
Table 12: Results of “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification.....	41
Table 13: Dataset Numbers for “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification.....	42
Table 14: Results of “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification	42
Table 15: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote” Classification	43
Table 16: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote” Classification	43
Table 17: Dataset Numbers for “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote” Classification	45
Table 18: Results of “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote” Classification.....	45
Table 19: Dataset Numbers for “NoProxy” vs “Proxy-Local” vs “Proxy-Remote” Classification.....	46
Table 20: Results of “NoProxy” vs “Proxy-Local” vs “Proxy-Remote” Classification	46
Table 21: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache” Classification	47
Table 22: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache” Classification	47
Table 23: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” Classification	49
Table 24: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” Classification	49
Table 25: Dataset Numbers for “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” Classification.....	50
Table 26: Results of “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” Classification ..	50
Table 27: Features employed in the trained models of the classifiers	52
Table 28: Confusion matrix for the prototype	60
Table 29: Performance of the prototype proxy traffic analysis system.....	60

INTRODUCTION

A proxy server is a host, which intercepts the network traffic in order to manipulate some of its properties. For example, most commonly known proxy is a web caching proxy that is originally invented to enhance the performance of web browsing by intercepting the traffic to check whether the requested web object is on the proxy cache or not.

However, today proxies are used also to meet the need for anonymous web surfing [1]. In this case, users can anonymously surf the web without revealing their own IP (Internet Protocol) addresses by using a proxy server as a stepping-stone. Technically, a user's online activity goes to the proxy first, which handles and sends along the user's requests for information, data, files, email, etc. In each case, the user's actual IP address is hidden once it goes through a proxy. Actually, most of the times when a proxy server retrieves information (objects) from web sites, it provides only its own identity to the sites visited. In this way, users connections look as if they are targeting the proxy server rather than the services they request. This feature of proxy servers is very advantageous to users especially when they are forced to use stepping stones in order to access Internet services that are blocked by their governments, service providers or organizations. Therefore, when proxy servers are used in such situations then the address based censorship mechanisms would fail.

Having said this, while web surfing over a proxy is an effective way to protect one's anonymity and freedom of speech, it is also like a double-sided sword, it may raise security problems, too [1]. In other words, attackers can use it to hide their anonymity as well! Under such a scheme, users are no longer accountable because their identity from the server's perspective is not trustworthy. Normally, a server identifies a user (client) by its IP address. However, any user can easily access a server via an unaccountable proxy, not to mention

malicious users. Therefore, a server can no longer assure whether the IP address associated with a connection is actually the address of a client or that of a stepping-stone, i.e. proxy. Moreover, in the context of malicious users, the usage of a proxy is usually associated with botnets, which have become a common infrastructure for cyber threats and online crime. On one hand, bots are often offered or sold as proxies to anyone who does not want to be traced for their activities on the Internet. On the other hand, because the use of proxies increases the difficulty to trace the originator, they can be used by bots or by regular users.

The major challenge to the above problems lies in the lack of the capability to unambiguously identify the originator of a web request, i.e. the originator of the traffic. When a server receives a request such as a HTTP (HyperText Transfer Protocol) request from a host (client machine), there is no systematic way to determine whether the host itself generates the request, or it is relaying the request for another host. To the best of our knowledge, there is no general method available for detecting the use of proxies from the server's perspective.

Therefore in this research, we study and evaluate a machine learning based approach on different types of data sets to understand how far we can push this approach to identify the incoming proxy base traffic on the server side. The objective of this research includes identifying proxies and the clients behind them based on their behaviour from the traffic log files that are on the network that is outside of the proxy firewall. To this end, by using a machine learning based approach we employ a holistic approach without looking into the content of the traffic and without checking a static feature such as trying to block proxy traffic using known proxy IP addresses or proxy identifiers in web log files. To achieve this, we (i) employ a mixture of log files to represent real-life proxy behavior, and we (ii) design and develop a data driven machine learning based approach to provide recommendations for the automatic identification of traffic

from an anonymous proxy. Finally, we investigate all of the above under both encrypted and non-encrypted traffic conditions. Our results are very promising in terms of identifying traffic systematically coming from proxies.

1. BACKGROUND

Most proxy servers simply allow a user to surf web sites on the Internet without having the user's browser type, IP address and other header information sent to the website that the user is viewing. It simply means that the web server of the website does not receive such information because the proxy server blocks it. So in the end, the servers that a user visits will not be able to determine the properties of the user's host such as its IP address etc. They may only know the proxy server was there, which raises another area of caution. Some sites may deny access to servers via known proxy servers.

As discussed earlier, a web proxy server, hereafter will be refereed as a proxy, serves the user's requests by connecting directly to the source (the original site that has the requested information) or by serving it from a cache, a copy of the requested information stored on the proxy because the information is requested often by many users. Web proxy caching is a way to store requested Internet objects (e.g. data like web pages) available via the HTTP protocol on a system closer to the requesting site. Web browsers can then use the local HTTP proxy cache, reducing access time as well as bandwidth consumption. This is often useful for Internet service providers or other organizations where many users share the bandwidth, because it enables the organization to increase timely delivery of information to its users.

Proxies come in different varieties:

- Transparent proxy: This type of proxy identifies itself as a proxy to the visited server. Moreover, it reveals the user's IP address, so it will not hide the user's identity.
- Anonymous Proxy: This type of proxy identifies itself as a proxy server. It is detectable (as a proxy), but provides reasonable anonymity for most users by hiding their IP addresses.
- Distorting Proxy: This type of proxy identifies itself as a proxy server, but creates an "incorrect" originating IP address available through the "HTTP" headers. So it provides anonymity by creating a false "identity".
- High-Anonymity Proxy: This type of proxy does not identify itself as a proxy server and does not reveal the original IP address of a user.

In this research, we are going to look into high-anonymity proxies in more detail since they are the most challenging ones to identify in network traffic logs.

2. SQUID PROXY SERVER

Squid is a caching proxy [2] for the Web supporting HTTP, HTTPS, FTP, and more. It reduces bandwidth and improves response times by caching and reusing frequently-requested web pages. Squid has extensive access controls and makes a great server accelerator. It runs on most available operating systems, including Windows and is licensed under the GNU GPL.

Squid was originally developed as the Harvest object cache, part of the Harvest project at the University of Colorado at Boulder. Further work on the program was completed at the University of California, San Diego and funded via two grants from the National Science Foundation. Duane Wessels forked the "last pre-commercial version of Harvest" and renamed it to Squid to avoid confusion with the commercial fork called Cached 2.0, which became NetCache. Squid

version 1.0.0 was released in July 1996. Squid is now developed almost exclusively through volunteer efforts.

Currently Squid is used by hundreds of Internet Providers worldwide to provide their users with the best possible web access. Squid optimizes the data flow between client and server to improve performance and caches frequently-used content to save bandwidth. Squid can also route content requests to servers in a wide variety of ways to build cache server hierarchies that optimize network throughput. Thousands of web sites around the Internet use Squid to improve their content delivery. Squid can reduce the server load and improve delivery speeds to clients. Squid can also be used to deliver content from around the world - copying only the content being used, rather than inefficiently copying everything. Finally, Squid's advanced content routing configuration allows one to build content clusters to route and load balance requests via a variety of web servers.

The Squid system can run at a hit-rate of approximately 75%, effectively quadrupling the capacity of the Apache servers behind them. This becomes particularly noticeable when a large surge of traffic arrives directed to a particular page via a web link from another site, as the caching efficiency for that page can be nearly 100%.

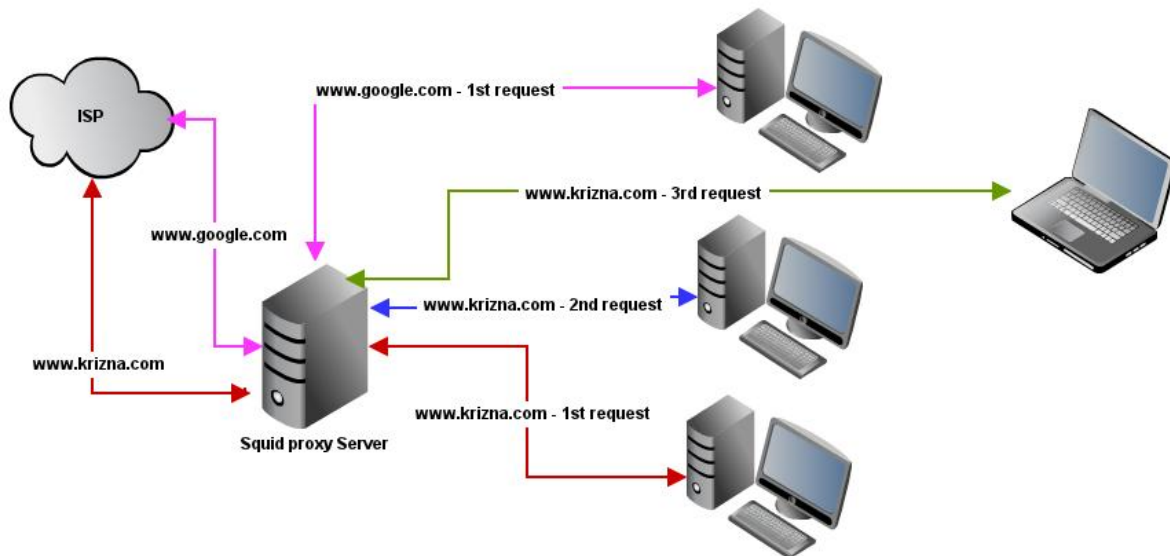


Figure 1: How Squid Proxy Server Works [2]

In the above figure, Figure 1, the Squid proxy caches the web content of *krizna.com* from the ISP (Internet Service Provider) during the first request and then it delivers the cached content for the further requests of *krizna.com* without requesting it from the original server. This will reduce bandwidth and will increase response time as the content is delivered from a local server, i.e. Squid proxy.

In this research, to create our proxy traffic data set, the Squid proxy server is chosen because Squid has some features that makes very suitable for researching on proxy traffic. First of all, Squid is a free and open source proxy server. Secondly, it is widely used by the Internet Service Providers (ISPs) all over the world. This enables the results of this research to potentially be used in practice. Finally, Squid can help anonymize connections, such as disabling or changing specific header fields in a client's HTTP requests. Last but not the least, the Squid proxy server can also be configured as a *high anonymity proxy* to not identify itself as a proxy server, so that

the web servers cannot recognize (under normal conditions) that the traffic is coming from a proxy server!

3. GENERATING PROXY DATA SETS

To generate our proxy dataset, we have set up the following network, Figure 2:

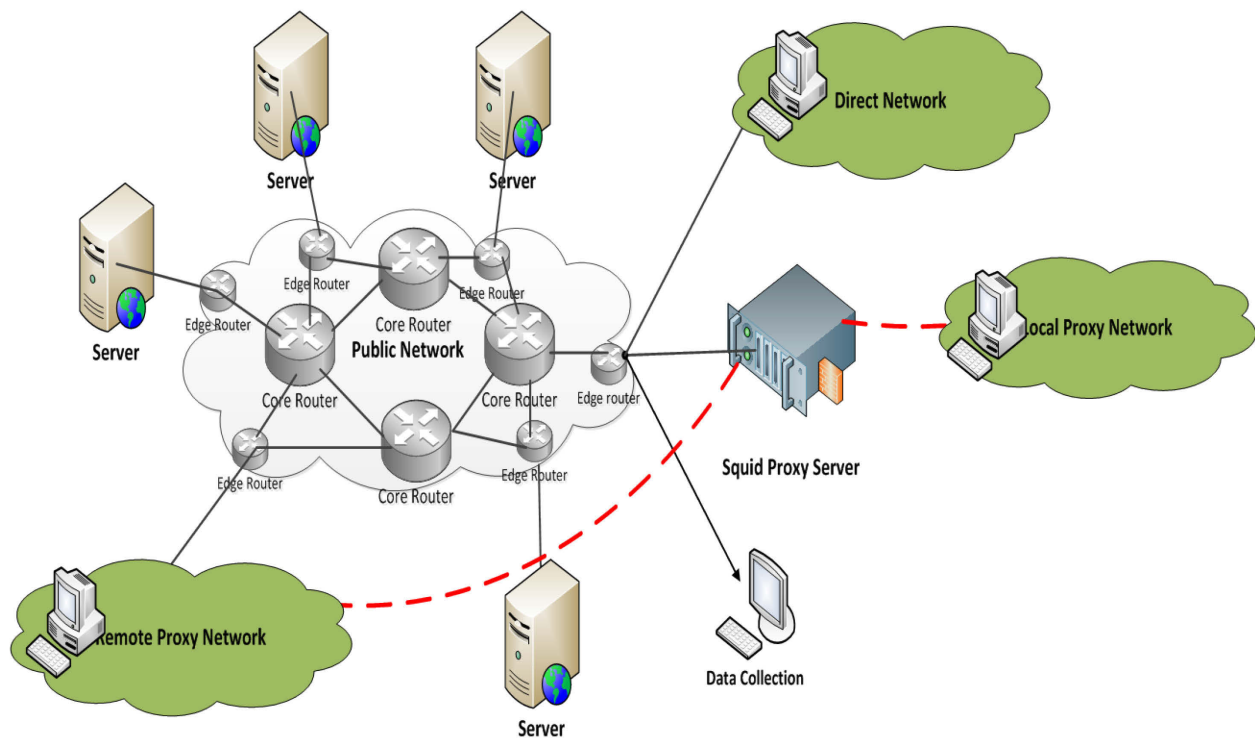


Figure 2: Our Testbed Network for Generating the Proxy traffic

As you can see in the figure, to generate our data sets, we have created three separate networks. These are:

- **Local Proxy Network:** This network is located in the Dalhousie University and directly connected to the Proxy server. The only way that this network can access to the Internet is to go through the Proxy server.

- Remote Proxy Network: This network is located outside the Dalhousie University, and connected to the Proxy server through the Internet. This network is configured to forward all of its traffic through the proxy server.
- Direct network: This network is directly connected to the Internet without using a proxy server.

To generate our data sets, we have used 500 website URLs that is provided by Alexa. This URL list, Appendix A, is then used to generate HTTP requests under different scenarios using the different proxy networks described above.

In the first scenario, all the HTTP requests to 500 Alexa websites are generated on the direct network and captured at the edge router. This traffic can be used to investigate the behaviors of the normal unencrypted HTTP traffic in our research.

Then, we implemented two scenarios for generating proxy traffic. In this case, (i) the proxy and the client can be both on the same network, we call this local proxy; or (ii) the proxy and the client could be on different networks, we call this remote proxy. In these two scenarios, namely local proxy and remote proxy, the proxy traffic generation process is repeated several times, each time with a different configuration mode of the Squid proxy server to understand how these configurations could affect the identification of proxy traffic. Different modes of the Squid proxy server include:

- Configuring the proxy server in the No-Cache mode, so that the proxy server just relays the traffic between the users and the web servers.
- Configuring the proxy server in the cache mode. In this case we, have generated two datasets:

- The cache server is empty, so that the proxy server has to refer to the web servers for every single request by the users, but at the same time it caches the traffic.
- The cache server has already cached the requests to the 500 web sites, so at this time the proxy server can response some of users' requests from its own cache. In this case, the generated traffic is lower than the previous mode.

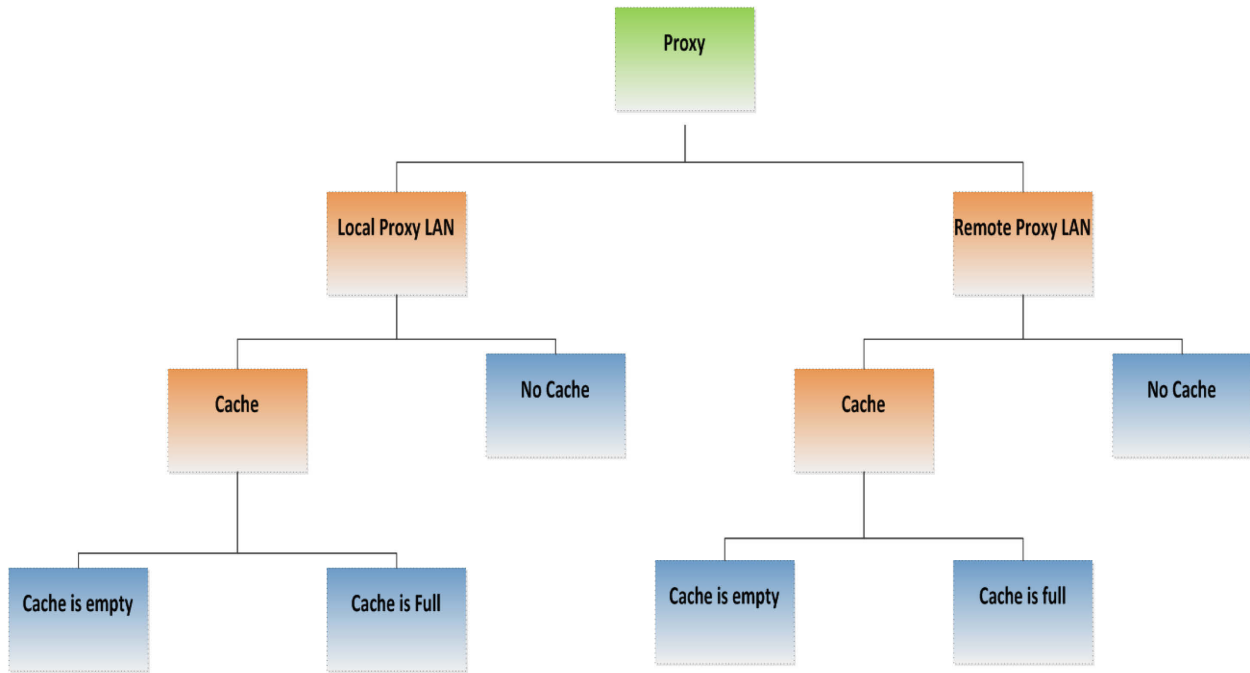


Figure 3: The Categories of the Generated Unencrypted Proxy Datasets in the NIMS Lab.

Figure 3 presents the category of each generated data sets. The 6 blue boxes indicate the 6 different generated proxy data sets. Once we generated these 6 proxy data sets, we have found that the default configuration of the Squid proxy server is in the transparent proxy mode. It means that the Squid proxy server embeds all the information about the users in the http user agents. To understand what exactly Squid proxy reveals in this mode, we use the following web site:

URL-1: <http://pgl.yoyo.org/http/browser-headers.php>

Figure 4 shows the information revealed about the client who accesses the above URL on the direct network we set up (without a proxy).

HTTP headers supplied by your browser (vahid-pc.research.cs.dal.ca):	
Host:	pgl.yoyo.org
Connection:	keep-alive
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent:	Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
Accept-Encoding:	gzip,deflate,sdch
Accept-Language:	en-US,en;q=0.8,fa;q=0.6

Figure 4: A Sample of the http User Agent without Using Proxy

Figure 5 shows the information revealed about the client who accesses the above URL via Local or Remote Proxy, where Squid Proxy is running in the transparent mode.

HTTP headers supplied by your browser (cisco-router-nims.research.cs.dal.ca):	
Host:	pgl.yoyo.org
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent:	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.102 Safari/537.36
Accept-Encoding:	gzip,deflate,sdch
Accept-Language:	en-US,en;q=0.8,fa;q=0.6
Via:	1.1 localhost (squid/3.1.19)
X-Forwarded-For:	192.168.100.6
Cache-Control:	max-age=0
Connection:	keep-alive

Figure 5: A Sample of the http User Agent in a Proxy Network with Default Configuration

As one can see in the above figure, when the proxy server is in the transport mode, it sends all the information of its client(s) to the web server that the client is accessing via the proxy. This enables the web server to infer that the traffic is coming from a host (client computer) behind the proxy server. In this case, the web server receives from the proxy running in the transparent mode also the local IP address of the client accessing its services. In the example given above, Figure 5, the local IP address of the client behind the proxy is: 192.168.100.6

There is also another well known website that shows if a user is currently behind a proxy server or not. Its URL is the following:

URL-2: <http://www.whatismyip.com/>

If a user visits this website at URL-2 from a network, which is not behind a proxy/firewall, in other words something like the direct network scenario, then the user should see something similar to Figure 6:

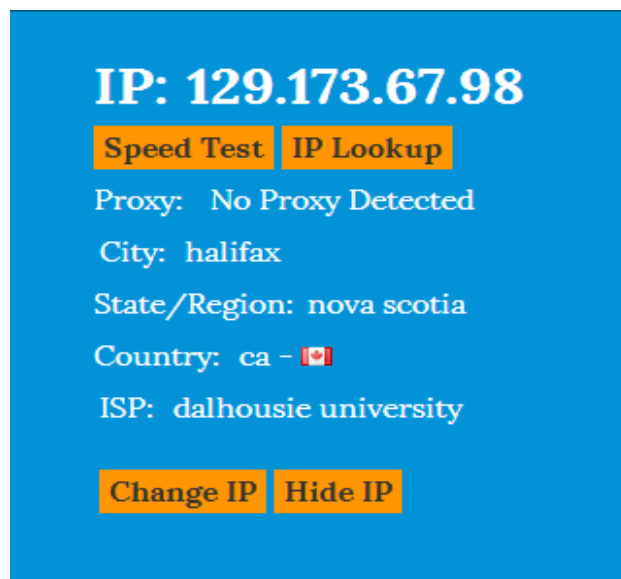


Figure 6: Visiting whatismyip Website (URL-2) on a Direct Network

But if a user visits this website from a computer in the local or remote proxy network, i.e. a network behind a proxy/firewall, then one can see something similar to Figure 7. However, this can be seen only if the proxy is working in the transparent mode.

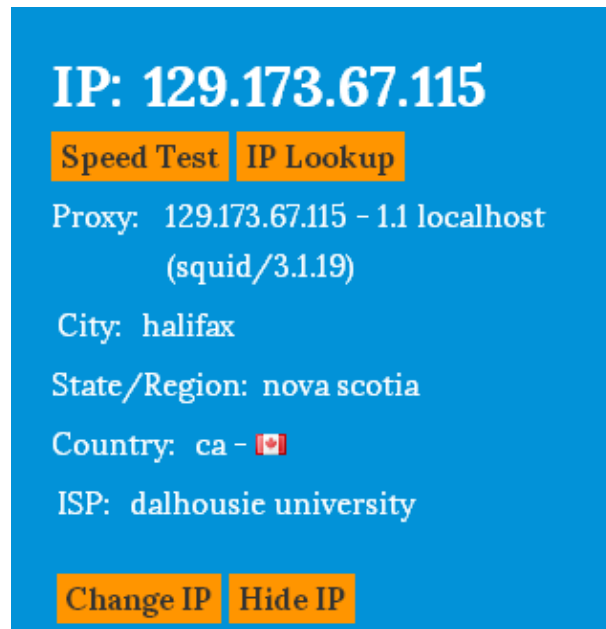


Figure 7: Visiting whatismyip Website (URL-2) on a Proxy Network with Default Configuration

As you can see, once the proxy server is setup (running) in the transparent mode, the web servers are able to infer that the source of the traffic is somewhere behind a proxy server. In our example, the IP address of the proxy server is 129.173.67.98.

We can also analyze the above scenario's http user agent by using the Wireshark [3] protocol analysis tool, Figure 8.

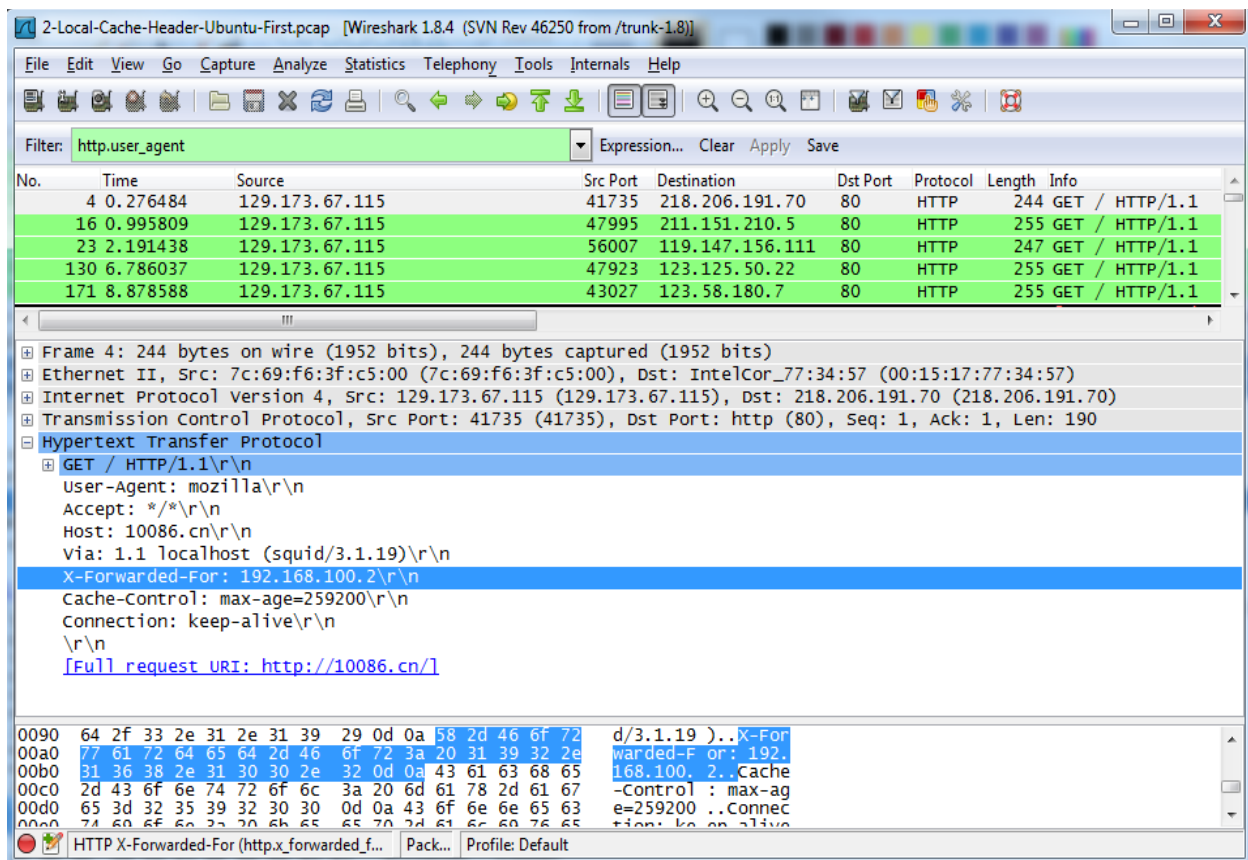


Figure 8: Viewing at http User Agent on Wireshark

As you can see in the Wireshark, Figure 8, the client IP address behind the proxy server (192.168.100.2), and also the name of the proxy server (Squid 3.1.19) can be seen in the traffic that is sent to the web server visited.

However, finding the existence of a proxy server and the computers behind that is not always this simple. As discussed before, there are four operation (configuration) modes of proxy servers: (i) Transparent mode, (ii) Anonymous mode, (iii) Distorting mode, and (iv) High Anonymity mode.

The aforementioned examples are all in the transparent mode. A clever user/attacker may use other modes of a proxy server to hide his/her identity. The anonymous and the distorting modes

provide some levels of anonymity, but the highest anonymity mode of a proxy server is the 4th one, High Anonymity mode. In this mode, not only the identity of the proxy cliets would not be sent to the web server(s) visited, but also the visited web server could not recognize (under normal conditions) that it is communicating with a proxy server. In this case the user/attacker can completely hide his/her identity from the web server. This mode is the most ambiguous mode from the perspective of the server that is visited (giving the service requested).

We configured the Squid proxy server to operate at the high anonymity mode by adding the following access controls to the Squid proxy configuration:

```
forwarded_for off  
request_header_access Allow allow all  
request_header_access Authorization allow all  
request_header_access WWW-Authenticate allow all  
request_header_access Proxy-Authorization allow all  
request_header_access Proxy-Authenticate allow all  
request_header_access Cache-Control allow all  
request_header_access Content-Encoding allow all  
request_header_access Content-Length allow all  
request_header_access Content-Type allow all  
request_header_access Date allow all  
request_header_access Expires allow all  
request_header_access Host allow all  
request_header_access If-Modified-Since allow all  
request_header_access Last-Modified allow all  
request_header_access Location allow all  
request_header_access Pragma allow all  
request_header_access Accept allow all
```


request_header_access Accept-Charset allow all
request_header_access Accept-Encoding allow all
request_header_access Accept-Language allow all
request_header_access Content-Language allow all
request_header_access Mime-Version allow all
request_header_access Retry-After allow all
request_header_access Title allow all
request_header_access Connection allow all
request_header_access Proxy-Connection allow all
request_header_access User-Agent allow all
request_header_access Cookie allow all
request_header_access All deny all

Then we again generated all the six proxy datasets explained before with the new proxy server configurations. Figure 9 shows the HTTP user agent in the proxy high anonymity mode when URL-1 is contacted. As you can see, there is no information revealed about the proxy server and the client(s) behind it, i.e. clients using it.

HTTP headers supplied by your browser (cisco-router-nims.research.cs.dal.ca)	
Host:	pgl.yoyo.org
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent:	Mozilla/5.0 (Windows NT 5.1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.102 Safari/537.36
Accept-Encoding:	gzip,deflate,sdch
Accept-Language:	en-US,en;q=0.8,fa;q=0.6
Connection:	keep-alive

Figure 9: Hiding the Proxy Server information in the http User Agent

Figure 10 shows the revealed information when URL-2 is contacted. As you can see in this figure, again no proxy information is revealed. The IP address of the client behind the proxy server is also completely hidden. This is what the (web) servers worry about the most, because there is no obvious way to find that this traffic is coming from a proxy.

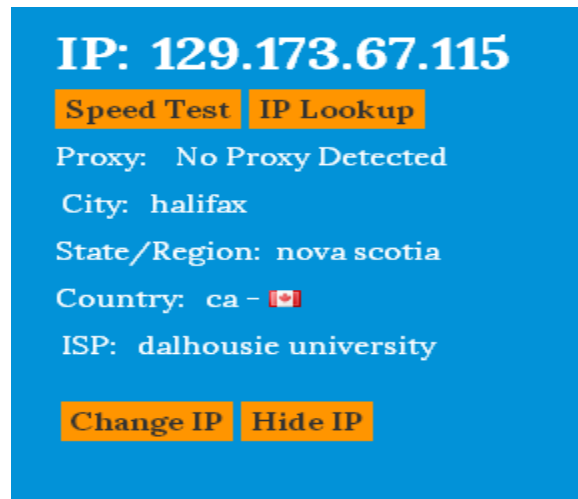


Figure 10: Visiting whatismyip Website When the Proxy Server is Configured to Hide Itself

In Table 1, we have summarized the information of all the 13 generated data sets.

Table 1: Summary of the Generated Unencrypted Proxy Datasets in the NIMS Lab

					Size (bytes)	#Packets	Duration (HH:MM)
No Proxy					80,747,884	96484	1:33
Proxy	Header	Local	Cache	Empty	80,616,017	100956	1:15
				Full	61,014,664	77242	1:05
		No cache			81,077,862	95305	1:02
		Remote	Cache	Empty	71,855,827	89135	1:07
				Full	51,100,296	62702	1:44
			No cache			69,429,793	84440
	No Header	Local	Cache	Empty	80,935,673	97891	1:15
				Full	58,901,656	70714	0:52
			No cache			79,659,292	93854
		Remote	Cache	Empty	66,834,471	79906	1:07
				Full	51,607,605	62691	0:56
No cache			66,265,355	79644	1:05		

Furthermore, Squid can also serve as a proxy for HTTPS (secure HTTP) traffic. However, in HTTPS, because the traffic is encrypted, it is not possible to cache HTTPS traffic. In HTTPS traffic, the whole communication between the client and the server is encrypted. Thus, in this case, a proxy intercepts the traffic between the server and the client, and just changes the port numbers and the IP addresses. There is also no way to change the user agent by the proxy, because everything in an HTTPS communication is encrypted. So, web caching and HTTP user agent configurations do not apply for encrypted proxy traffic.

Because Alexa does not provide a list of web sites only using HTTPS as their communication protocol, we created our own list of web sites, Appendix B. This list contains 176 URLs of web sites using HTTPS as their communication protocol.

To generate our HTTPS (encrypted traffic) proxy datasets, we set up the use of the same network setup as we did for generating the HTTP (unencrypted traffic) proxy datasets. To this end, first of all, we run the HTTPS web requests to all the web sites (176) on the list on the direct network and captured the resulting traffic at the edge router. This traffic is used to investigate the behaviors of the normal HTTPS (encrypted HTTP) traffic in our research. Then, we repeated this process for the clients on the local proxy network and the remote proxy network. For generating the HTTPS proxy data sets, we did not need to reconfigure the Squid sever, because Squid cannot cache, read or change the encrypted traffic. In Table 2, we have summarized the information of our HTTPS (encrypted) proxy data sets.

Table 2: Summary of the Generated Encrypted Proxy Datasets in the NIMS Lab

	Size (bytes)	#Packets	Netmate	Tranalyzer	Duration
No Proxy	135,702,128	153821	1278	2621	1:26
Local Proxy Network	166,926,162	245598	6438	12896	2:17
Remote Proxy Network	216,829,519	309518	7946	15895	2:31

4. STATE OF THE ART METHODS FOR DETECTING PROXY TRAFFIC

In the literature, there are not many studies that aim to identify proxy traffic on a server that is outside of the proxy/firewall network [4 – 10]. Even then, the few that exist [5, 10] require some information about either the proxy or the client behind the proxy. In all cases, we can group the schemes used into two general categories:

- (i) Active measurement; and
- (ii) Passive measurement based schemes. In the rest of this section, we summarize these schemes and show their limitations.

4.1. Active Measurement Based Schemes

Under these schemes, it is assumed that the traffic generated by a regular client (without going through a proxy) would have different behaviour compared with the traffic relayed by a proxy that a client uses to forward his/her packets. Some researchers in the field have used this assumption by developing different schemes to detect the presence of a proxy by observing the inter-arrival times and payload sizes of individual packets arriving at a server, such as a web server. Using such schemes different researchers [6, 7, 8, 9, 10] claimed to achieve approximately 90% in detecting proxy traffic.

In general these schemes employ additional packets, called “active probes”, to measure the inter-arrival times, or more generally the delays packets are facing on the network. So active probes are injected (sent) into the traffic network and their transit times are used to estimate (sample) the network delay, namely RTT (Round Trip time), on the path the probes follow at the time they are sent. The RTT of a path is calculated by summing up the actively measured delays

in each direction of the path. For example, ping command is the used as one of the active measurement schemes to calculate RTTs.

One of the major problems of this scheme is that the client should be configured to reply to the active probes, i.e. measurement packets, send from the host that is analyzing the RTTs. For example, if a web server is analyzing the traffic, then the server will send the ICMP (Internet Control Message Protocol) Ping packets, or any “active probe” scheme implemented. However, such a scheme would only work if the client replies to these probing requests. Another challenge of these schemes is that most of the times routers handle ICMP packets or active known probing packets in their slow path (leading to overestimation of RTT), or they simply discard them. Furthermore, most of the proxy servers with default configurations discard such ICMP probing packets as well. In summary, such schemes are irrelevant for the type of analysis we are doing in this research, because we require we do the analysis of the traffic outside of the proxy/firewall and we do not have access or a priori information either on the proxy or on the client using the proxy.

4.2. Passive Measurement Based Schemes

These schemes [4, 5] also make use of the assumption regarding the delays experienced and using them to identify proxies, but some passive schemes also enhance passive delay measurements with other information such as operating system or web browser information, again passive measured or fingerprinted. Passive schemes do not introduce additional packets, i.e. active probe packets, onto the network. Instead, they make use of the existing information in the traffic captures or other log files. We summarized the well-known passive measurement techniques below.

4.2.1. OWD – One Way Delay

This technique measures one-way delay by noting the time it takes an arbitrary packet to transit between two precisely synchronized measurement points. The major limitation of this technique is that the OWD needs to be set up in several measurement points along the path, and also the time between these measurement points need to be synchronized. This does not meet our requirement that is we only have access to the servers, where the analysis is made, but not the other nodes on the path (network).

4.2.2. Single Measuring Point

In this case, the RTT is calculated from the time between a request packet being seen heading towards a distant server, and a matching reply packet coming back from the same server. Request/response packet-pairs are matched based on well-known fields in the packet header or payload (e.g. sequence numbers in TCP or ICMP echo packets). One of the major limitations of this approach is that it requires measurements on the client machine. Thus, this scheme works when we are at the position of a client machine and want to calculate the RTT to the server, but not when we are at the position of the web server and want to calculate the RTT to the client. Therefore, it does not meet our requirement.

4.2.3. SPP – Synthetic Packet Pairs

This technique, SPP, estimates the RTT between two measurement points along a network path. Traffic is observed at both measurement points, and the RTT between the two measurement points is estimated from pairs of packets seen travelling in each direction. Again, the main limitation of this technique is that it requires traffic traces from both of the server and the client sides. Therefore, this does not meet our requirement either.

4.2.4. TTL – Time To Live

This technique aims to infer the presence of a proxy (in the form of a network address translation device – NAT) based on the TTL values of the packets sent by clients (IP addresses) and captured on the server. It is assumed that if the TTL is $\text{ttl}_{\text{init}} - 1$, the sending host is directly connected to the Internet (as the monitoring point is one hop away from the device on which the traffic is monitored). If the TTL is $\text{ttl}_{\text{init}} - 2$ then there is a routing device such as a NAT or a proxy in the users' premises. Indeed, this assumes that the number of hops between the machine that the traffic is captured and the machine where the analysis is made is known. Only then the TTL values can be interpreted to detect a proxy. Moreover, one of the major limitations of this technique is that a proxy can reset the TTL value of the packets and sets its own TTL value. So, this technique will be irrelevant under such conditions.

4.2.5. TTL and OS fingerprinting

Some researchers extended the passive measurements into the HTTP user agent strings (when the information is available) to observe the OS types and their versions as well as browser information (type and version). In this case, the assumption is that it is possible to detect a proxy more accurately based on the OS and/or browser fingerprint. Mostly, HTTP user agent string is used to do the fingerprinting of OSs and/or web browsers. Nevertheless, this technique has limitations, too. Similar to the previous techniques, if a proxy is left by its default TTL configuration, then this system cannot infer the presence of a Proxy device. If all the hosts behind a Proxy network use the same type of OS and/or browser, this technique cannot detect the proxy traffic. Moreover, any host that has more than one OS and/or browser installed would be considered a proxy under this scheme. Last but not the least, this technique will not be any

different than the previous techniques when the traffic is encrypted, because the HTTP user agent string will be opaque in encrypted traffic.

5. Machine Learning Based Approach

Given the limitations of the state of the art techniques discussed above, in this research, we propose a machine learning based approach to identify high level behaviour of proxy machines in a given network traffic trace. To this end, we have employed two classification based learning techniques to evaluate on our data sets. These are C4.5 decision tree classifier and the Naïve Bayes classifier.

Classification is a supervised learning technique, where the aim is to learn a mapping from the input space to the output space whose correct values (labels) are provided by a supervisor (ground truth, in other words real labels). Thus, both of the learning techniques employed in this work require a training phase to learn the patterns and/or mappings in the input data. Then the learned models are evaluated on unseen test data. The following summarizes the C4.5 Decision Tree and Naïve Bayes algorithms.

5.1. Decision Tree Algorithm

C4.5 is a decision tree based classification algorithm developed by Ross Quinlan that is an extension of the basic ID3 algorithm [11]. C4.5 is designed to address the following issues that are not performed in ID3 such as choosing the appropriate attribute (based on information gain), trying to reduce error pruning, and handling varieties of attributes types (continuous, number, string). It should be noted here that we use the words “attribute” and “feature” interchangeably in the rest of this report.

A decision tree is a hierarchical data structure for implementing a divide-and-conquer strategy. C4.5 is an efficient non-parametric method that can be used both for classification and regression. In non-parametric models, C4.5 constructs decision trees from a set of training data applying the concept of information entropy. The training data is a set, S , such that each input of the set is an instance of already classified samples. Each sample in the set is a vector where each input in the vector represents an attribute or feature of the sample. The training data is added to a vector where each input in the vector represents the class that each sample belongs to. C4.5 can split the data into smaller subsets using the fact that each attribute of the data can be used to make a decision. Therefore, the attribute with the highest information gain is used to make the decision of the split. As a result, the input space is divided into local regions defined by a distance metric. In a decision tree, the local region is identified in a sequence of recursive splits in small number of steps. A decision tree is composed of internal decision nodes and terminal leaves. Each node, m , implements a test function $f_m(x)$ with discrete outcomes labeling the branches. This process starts at the root and is repeated until a leaf node is hit. The value of a leaf constitutes the output. In the case of a decision tree for classification, the goodness of a split is quantified by an impurity measure. A split is pure if for all branches, for all instances choosing a branch belongs to the same class after the split. One possible function to measure impurity is entropy, Eq. (1) [12].

$$I_m = -\sum_{j=1}^n p_m^j \log_2 p_m^j \quad (1)$$

If the split is not pure, then the instances should be split to decrease impurity, and there are multiple possible attributes on which a split can be done. Indeed, this is locally optimal; hence there is no guarantee of finding the smallest decision tree. In this case, the total impurity after the

split can be measured by Eq. (2). In other words, when a tree is constructed, at each step the split that results in the largest decrease in impurity is chosen. This is the difference between the impurity of data reaching node m , Eq. (1), and the total entropy of data reaching its branches after the split, Eq. (2). Figure 11: Construction of a classification tree [12] presents the construction of a classification tree. A more detailed explanation of C4.5 algorithm can be found in [12].

$$I'_m = - \sum_{j=1}^n \frac{N_{mj}}{N_m} \sum_{i=1}^k p_{mj}^i \log p_{mj}^i \quad (2)$$

```

GenerateTree( $\mathcal{X}$ )
  If NodeEntropy( $\mathcal{X}$ ) <  $\theta_I$  /* equation 9.3 */
    Create leaf labelled by majority class in  $\mathcal{X}$ 
    Return
   $i \leftarrow \text{SplitAttribute}(\mathcal{X})$ 
  For each branch of  $x_i$ 
    Find  $\mathcal{X}_i$  falling in branch
    GenerateTree( $\mathcal{X}_i$ )

SplitAttribute( $\mathcal{X}$ )
  MinEnt  $\leftarrow$  MAX
  For all attributes  $i = 1, \dots, d$ 
    If  $x_i$  is discrete with  $n$  values
      Split  $\mathcal{X}$  into  $\mathcal{X}_1, \dots, \mathcal{X}_n$  by  $x_i$ 
       $e \leftarrow \text{SplitEntropy}(\mathcal{X}_1, \dots, \mathcal{X}_n)$  /* equation 9.8 */
      If  $e < \text{MinEnt}$  MinEnt  $\leftarrow e$ ; bestf  $\leftarrow i$ 
    Else /*  $x_i$  is numeric */
      For all possible splits
        Split  $\mathcal{X}$  into  $\mathcal{X}_1, \mathcal{X}_2$  on  $x_i$ 
         $e \leftarrow \text{SplitEntropy}(\mathcal{X}_1, \mathcal{X}_2)$ 
        If  $e < \text{MinEnt}$  MinEnt  $\leftarrow e$ ; bestf  $\leftarrow i$ 
  Return bestf

```

Figure 11: Construction of a classification tree [12]

5.2. Naïve Bayes Algorithm

A Naïve Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. In simple terms, a

naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature. Depending on the precise nature of the probability model, Naïve Bayes classifiers can be trained efficiently in a supervised learning approach. In many practical applications, parameter estimation for Naïve Bayes models uses the method of maximum likelihood [13]. A simple Naïve Bayes probabilistic model can be expressed as Eq. (3) in the following:

$$P(C|F_1, F_2, \dots, F_n) = \frac{1}{Z} P(C) \prod_{i=1}^n P(F_i|C), \quad (3)$$

where $P(C|F_1, F_2, \dots, F_n)$ is the probabilistic model over dependent class variable C with a small number of outcomes or classes, conditional on several feature variables F_1 through F_n ; Z is a scaling factor dependent only on F_1, F_2, \dots, F_n , i.e., a constant if the value of the feature variables are known. A Naïve Bayes classifier combines the probabilistic model with a decision rule that aims to maximize a posterior, thus the classifier can be defined using Eq. (4) as follows:

$$\text{Classify}(f_1, f_2, \dots, f_n) = \text{argmax}_c P(C = c) \prod_{i=1}^n P(F_i = f_i|C = c) \quad (4)$$

An advantage of the Naïve Bayes classifier is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification. Given that independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix. Time complexity for learning the Naïve Bayes classifier is $O(N_p)$, where N is the number of training examples and p is the number of features. Space complexity for the Naïve Bayes algorithm is $O(pqr)$, where p is the number of features, q is values for each features, and r is alternative values for the class. Naïve Bayes is the simplest form of a Bayesian network. All attributes are independent given the value of class variables. This is called conditional independence. The conditional independence assumption is

not often true in the real world problems. A more detailed explanation of Naïve Bayes algorithm can be found in [12].

6. EVALUATIONS

The evaluations conducted in this work have seven components. These are: (i) Flow generation and feature extraction; (ii) Classification of flows into “Proxy” vs “No-Proxy”; (iii) Classification of Proxy flows into “Encrypted” vs “Unencrypted”; (iv) Classification of Proxy flows into “Local” vs “Remote”; (v) Classification of Encrypted Proxy flows into “Header” vs “No-Header”; (vi) Classification of Encrypted Proxy flows into “Cache” vs “No-Cache”; and (vii) Graphical user interface for visualizing the high level application behavior. Figure 12 gives an overview of the prototype system developed and used in these evaluations.

In our prototype, all network traffic data sets are first converted to traffic flows using two open source tools, namely NetMate [14] and Tranalyzer [15]. In doing so, we aim to understand if the different features of these flow tools will effect the performance of the classifiers or not. Flows are bidirectional and the first packet seen by the tool determines the forward direction. We consider only UDP and TCP flows. Moreover, UDP flows are terminated by a flow timeout, whereas TCP flows are terminated upon proper connection teardown or by a flow timeout, whichever occurs first. The flow timeout value employed in this work is 600 seconds as recommended by the IETF [16]. Also, in this work, all the “broken” flows are filtered out so that what are left are only the flows that have at least one packet in each direction.

Next, the statistical features generated by the flow tools shown in Table 3 are extracted from these traffic flows. It should be noted here that detailed explanation of these features can be found on the web sites of NetMate [14] and Tranalyzer [15] tools, respectively. Once these

features are extracted, then they are used to represent the network flows to the machine learning algorithms employed in this work.

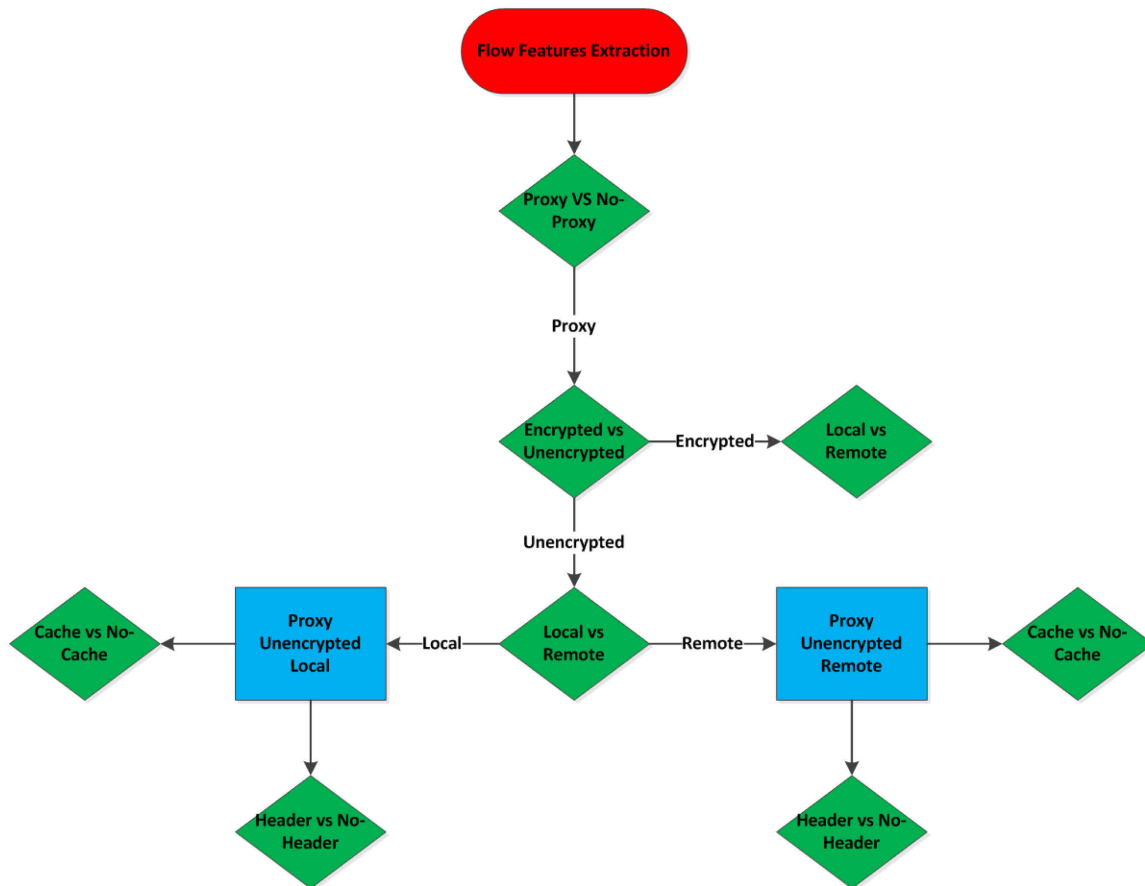


Figure 12: An Overview of Our Prototype System

Table 3: Features Employed in This Research

All Netmate Features	total_fpackets, total_fvolume, total_bpackets, total_bvolume, min_fpktl, mean_fpktl, max_fpktl, std_fpktl, min_bpktl, mean_bpktl, max_bpktl, std_bpktl, min_fiat, mean_fiat, max_fiat, std_fiat, min_biat, mean_biat, max_biat, std_biat, duration, min_active, mean_active, max_active, std_active, min_idle, mean_idle, max_idle, std_idle, sflow_fpackets, sflow_fbytes, sflow_bpackets, sflow_bbytes, fpsh_cnt, bpsh_cnt, furg_cnt, burg_cnt, total_fhlen, total_bhlen
All Tranalyzer Features	Direction, Duration, numPktsSnt, numPktsRcvd, numBytesSnt, numBytesRcvd, minPktSz, maxPktSz, avePktSize, pktps, bytps, pktAsm, bytAsm, ipMindIPID, ipMaxdIPID, ipMinTTL, ipMaxTTL, ipTTLChg, ipTOS, ipFlags, ipOptCnt, ipOptCpCl_Num, tcpPSeqCnt, tcpSeqSntBytes, tcpSeqFaultCnt, tcpPAckCnt, tcpFlwLssAckRcvdBytes, tcpAckFaultCnt, tcpInitWinSz, tcpAveWinSz, tcpMinWinSz, tcpMaxWinSz, tcpWinSzDwnCnt, tcpWinSzUpCnt, tcpWinSzChgDirCnt, tcpAggrFlags, tcpAggrAnomaly, tcpOptPktCnt, tcpOptCnt, tcpAggrOptions, tcpMSS, tcpWS, tcpS-SA/SA-ATrip, tcpRTTSseqAA, tcpRTTackTripMin, tcpRTTackTripMax, tcpRTTackTripAve

To this end, our 13 proxy traffic traces (discussed earlier) are employed for evaluation purposes. Brief statistics on these traffic traces are given in Table 1, Table 2, and Table 4

Once the traffic traces are represented using statistical features based on flows, the next step is to randomly sample (using uniform probability) data sets from the different categories of flows. In this work, the C4.5 and the Naïve Bayes classification algorithms are used to classify the Proxy traffic traces into Proxy vs No-Proxy, Encrypted vs Unencrypted, Local vs Remote, Header vs No-Header, and Cache vs No-Cache. It should be noted here that the open source tool WEKA [17] is employed for running the classifiers as well as for performing the random sampling using uniform probability distribution function.

In summary, any network traffic trace file that will be analyzed using this system, first needs to be converted into flows using a tool such as Netmate and the aforementioned features needs to be extracted. Then the prototype system classifies these flows, using a trained classification model, to classify Proxy traffic vs No-Proxy traffic. For this work, we employ the C4.5 and Naïve Bayes learning techniques to create such classification models. Then the output of the

aforementioned process becomes the input for the Encrypted Proxy vs Unencrypted Proxy classifier for identifying high level behavior of the proxy traffic. After this, all flows identified as proxy, encrypted and unencrypted, run through the Local Proxy vs Remote Proxy classifiers. This classifier detects whether the machines behind the proxy device are located in the same network as the proxy device is located, or located in a separate network. Whether the unencrypted proxy traffic is classified as local or remote, it runs through two other classifiers which are Cache vs No-Cache classifier and Header vs No-Header Classifier (this classifier detects if there is any fingerprint of the proxy device in the header of the http request or not). It should be noted here that all of this analysis (classification) is performed on a machine that is on a different network than the proxy and its clients.

7. EXPERIMENTS AND RESULTS

In this work, the learning models of the C4.5 and Naïve Bayes algorithms are trained and tested using WEKA [17]. As discussed earlier, aforementioned traffic data sets are used during these evaluations. The NIMS Lab proxy traffic data sets are available for testing and benchmarking purposes in the attached CD.

In traffic classification, two metrics are typically used in order to quantify the performance of the classifier: Detection Rate (DR) and False Positive Rate (FP). In this case, DR reflects the number of in-class (the class that we are interested in) flows correctly classified and is calculated using the $DR = TP/(TP+FN)$; whereas the FP rate reflects the number of out-class (anything that is not in-class) flows incorrectly classified as in-class using the $FPR = FP/(FP+TN)$. Naturally, a high DR rate and a low FP rate are the most desirable outcomes. Moreover, False Negative, FN, implies that in-class traffic is classified as out-class traffic, and False Positive, FP, implies that

out-class traffic is classified as in-class traffic. In the following we present the results of the C4.5 and Naïve Bayes classification algorithms on our proxy traffic data sets.

7.1. Results of the Classification Experiments

Given our approach discussed above, the first set of experiments we performed are training and testing of the C4.5 and Naïve Bayes classifiers for classifying the flows into “Proxy” vs “No-Proxy”, then “Encrypted Proxy” vs “Unencrypted Proxy”, then “Local Proxy” vs “Remote Proxy”, then “Encrypted Proxy with Header” vs “Encrypted Proxy with No-Header”, and finally, “Cached Encrypted Proxy” vs “No-Cached Encrypted Proxy”.

Table 4 assigns a number to each of our proxy traffic datasets for ease of further referencing, and also represents the number of extracted flows from each datasets, using Netmate and Tranalyzer tools.

To identify the proxy traffic, we considered 11 different cases, including:

1. “NoProxy-Unencrypted” vs “Proxy-Unencrypted”
2. “NoProxy-Encrypted” vs “Proxy-Encrypted”
3. “NoProxy” vs “Proxy”
4. “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted”
5. “Proxy-Encrypted” vs “Proxy-Unencrypted”
6. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote”
7. “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote”
8. “NoProxy” vs “Proxy-Local” vs “Proxy-Remote”

9. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache”
10. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader”
11. “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader”

For each case, we did 8 separate tests including the combination of Balance/Unbalance datasets, Netmate/Tranalyzer features, and C4.5/Naïve Bayes classification algorithms. Below we present the results of the entire 88 separate proxy traffic experiments performed in the scope of this work.

Table 4: The Number of Instances (flows) in our Datasets

Dataset Number	Dataset Name	Number of Netmate Instances	Number of Tranalyzer Instances
1	Unencrypted-NoProxy	1127	2323
2	Unencrypted-Remote-Cache-Header-EmptyCache	1136	2286
3	Unencrypted-Remote-Cache-Header-FullCache	1076	2166
4	Unencrypted-Remote-Cache-NoHeader-EmptyCache	1116	2247
5	Unencrypted-Remote-Cache-NoHeader-FullCache	1071	2158
6	Unencrypted-Remote-NoCache-Header	1111	2238
7	Unencrypted-Remote-NoCache-NoHeader	1110	2240
8	Unencrypted-Local-Cache-Header-EmptyCache	1251	2519
9	Unencrypted-Local-Cache-Header-FullCache	1134	2282
10	Unencrypted-Local-Cache-NoHeader-EmptyCache	1281	2578
11	Unencrypted-Local-Cache-NoHeader-FullCache	1143	2300
12	Unencrypted-Local-NoCache-Header	1252	2518
13	Unencrypted-Local-NoCache-NoHeader	1271	2556
14	Encrypted-NoProxy	1278	2556
15	Encrypted-Remote	7946	15894
16	Encrypted-Local	6438	12877

7.1.1. “NoProxy-Unencrypted” vs “Proxy-Unencrypted”: Binary Classification

In this case, our aim is to differentiate unencrypted traffic without any proxies from unencrypted traffic with proxies. It turns out this is the most challenging case we experimented with, 90% detection with almost 8% false alarm.

Table 5: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted” Classification

Class	Dataset Number
NoProxy-Unencrypted	1
Proxy-Unencrypted	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Table 6: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted: 13952	C4.5	DR	FPR
			NoProxy-Unencrypted	74.4	1.4
			Proxy-Unencrypted	98.6	25.6
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	74.9	23.9
			Proxy-Unencrypted	76.1	25.1
	Balance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted: 1127	C4.5	DR	FPR
			NoProxy-Unencrypted	92.5	10.8
			Proxy-Unencrypted	89.2	7.5
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	88.6	33.5
			Proxy-Unencrypted	66.5	11.4
Tranalyzer	Unbalance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted: 28088	C4.5	DR	FPR
			NoProxy-Unencrypted	73.1	0.7
			Proxy-Unencrypted	99.3	26.9
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	33.8	10.6
			Proxy-Unencrypted	89.4	66.2
	Balance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted: 2323	C4.5	DR	FPR
			NoProxy-Unencrypted	89.6	10
			Proxy-Unencrypted	90	10.4
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	27.2	8.5
			Proxy-Unencrypted	91.5	72.8

7.1.2. “NoProxy-Encrypted” vs “Proxy-Encrypted”: Binary Classification

In this case, our aim is to differentiate encrypted traffic without any proxies from encrypted traffic with proxies. It turns out that this we can do this with very high performance 96% detection and 2% false alarm.

Table 7: Dataset Numbers for “NoProxy-Encrypted” vs “Proxy-Encrypted” Classification

Class	Dataset Number
NoProxy-Encrypted	14
Proxy-Encrypted	15, 16

Table 8: Results of “NoProxy-Encrypted” vs “Proxy-Encrypted” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy-Encrypted: 1278 Proxy-Encrypted: 14384	C4.5	DR	FPR
			NoProxy-Encrypted	94	0.3
			Proxy-Encrypted	99.7	6
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	13.2	2.5
			Proxy-Encrypted	97.5	86.8
	Balance	NoProxy-Encrypted: 1278 Proxy-Encrypted: 1278	C4.5	DR	FPR
			NoProxy-Encrypted	97.7	3.8
			Proxy-Encrypted	96.2	2.3
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	16	2
			Proxy-Encrypted	98	84
Tranalyzer	Unbalance	NoProxy-Encrypted: 2556 Proxy-Encrypted: 28771	C4.5	DR	FPR
			NoProxy-Encrypted	91.6	0.4
			Proxy-Encrypted	99.6	8.4
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	17.2	3.7
			Proxy-Encrypted	96.3	82.8
	Balance	NoProxy-Encrypted: 2556 Proxy-Encrypted: 2556	C4.5	DR	FPR
			NoProxy-Encrypted	95.9	3.8
			Proxy-Encrypted	96.2	4.1
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	19.8	4.8
			Proxy-Encrypted	95.2	80.2

7.1.3. “NoProxy” vs “Proxy”: Binary Classification

In this case, our aim is to differentiate traffic without any proxies from traffic with proxies. This is the most complex case because it includes both the encrypted as well as the unencrypted traffic. We had assumed this would be the most challenging case, but our performance is pretty good (given the traffic is not filtered at all and includes everything) with 92% detection and 6% false alarm.

Table 9: Dataset Numbers for “NoProxy” vs “Proxy” Classification

Class	Dataset Number
NoProxy	1, 14
Proxy	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16

Table 10: Results of “NoProxy” vs “Proxy” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy: 2405 Proxy: 28336	C4.5	DR	FPR
			NoProxy	82.5	1
			Proxy	99	17.5
			Naïve Bayes	DR	FPR
			NoProxy	17.3	4.7
			Proxy	95.3	82.7
	Balance	NoProxy: 2405 Proxy: 2405	C4.5	DR	FPR
			NoProxy	94.1	7.9
			Proxy	92.1	5.9
			Naïve Bayes	DR	FPR
			NoProxy	17.9	4.6
			Proxy	95.4	82.1
Tranalyzer	Unbalance	NoProxy: 4879 Proxy: 56859	C4.5	DR	FPR
			NoProxy	80.6	0.7
			Proxy	99.3	19.4
			Naïve Bayes	DR	FPR
			NoProxy	34	6.3
			Proxy	93.7	66
	Balance	NoProxy: 4879 Proxy: 4879	C4.5	DR	FPR
			NoProxy	93.4	6.7
			Proxy	93.3	6.6

			<table><tr><td>Naïve Bayes</td><td>DR</td><td>FPR</td></tr><tr><td>NoProxy</td><td>36.5</td><td>7.7</td></tr><tr><td>Proxy</td><td>92.3</td><td>63.5</td></tr></table>	Naïve Bayes	DR	FPR	NoProxy	36.5	7.7	Proxy	92.3	63.5
Naïve Bayes	DR	FPR										
NoProxy	36.5	7.7										
Proxy	92.3	63.5										

7.1.4. “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted”: 3 Classes Classification

In this case, our aim is to differentiate traffic without any proxies from unencrypted proxy traffic as well as encrypted proxy traffic. Our performance is very good with over 90% detection and less than 5% false alarm.

Table 11: Dataset Numbers for “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification

Class	Dataset Number
NoProxy	1, 14
Proxy-Encrypted	15, 16
Proxy-Unencrypted	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Table 12: Results of “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy: 2405 Proxy-Encrypted: 14384 Proxy-Unencrypted: 13952	C4.5	DR	FPR
			NoProxy	82.7	0.9
			Proxy-Encrypted	99.4	0.8
			Proxy-Unencrypted	98.3	2
			Naïve Bayes		
			Naïve Bayes	DR	FPR
			NoProxy	16.7	4.3
			Proxy-Encrypted	96.1	28.4
	Proxy-Unencrypted	72.8	2.9		
	Balance	NoProxy: 2405 Proxy-Encrypted: 2405 Proxy-Unencrypted: 2405	C4.5	DR	FPR
			NoProxy	90.5	4.3
			Proxy-Encrypted	97.3	1.6
			Proxy-Unencrypted	93.5	3.5
			Naïve Bayes		
Naïve Bayes			DR	FPR	
NoProxy			17.2	4.4	
Proxy-Encrypted			95.6	43.6	
Proxy-Unencrypted	74.3	8.5			
Tranalyzer	Unbalance	NoProxy: 4879 Proxy-Encrypted: 28771 Proxy-Unencrypted: 280888	C4.5	DR	FPR
			NoProxy	82.5	0.7
			Proxy-Encrypted	99.4	0.9
			Proxy-Unencrypted	98.8	2

Balance			Naïve Bayes	DR	FPR
			NoProxy	11.8	3
			Proxy-Encrypted	96.6	43.8
			Proxy-Unencrypted	57.6	3.1
	NoProxy: 4879 Proxy-Encrypted: 4879 Proxy-Unencrypted: 4879	C4.5	DR	FPR	
			NoProxy	90.3	3.8
			Proxy-Encrypted	97.3	1.9
			Proxy-Unencrypted	94	3.4
		Naïve Bayes	DR	FPR	
			NoProxy	13	3.3
			Proxy-Encrypted	96.5	57.7
			Proxy-Unencrypted	55.5	6.5

7.1.5. “Proxy-Encrypted” vs “Proxy-Unencrypted”: Binary Classification

In this case, our aim is to differentiate encrypted proxy traffic from unencrypted proxy traffic. Our performance is very high in this case with over 99% detection and less than 0.5% false alarm.

Table 13: Dataset Numbers for “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification

Class	Dataset Number
Proxy-Encrypted	15, 16
Proxy-Unencrypted	2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

Table 14: Results of “Proxy-Encrypted” vs “Proxy-Unencrypted” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	Proxy-Encrypted: 14384 Proxy-Unencrypted: 13952	C4.5	DR	FPR
			Proxy-Encrypted	99.8	0.2
			Proxy-Unencrypted	99.8	0.2
			Naïve Bayes	DR	FPR
			Proxy-Encrypted	96.4	21.6
			Proxy-Unencrypted	78.4	3.6
	Balance	Proxy-Encrypted: 13952 Proxy-Unencrypted: 13952	C4.5	DR	FPR
			Proxy-Encrypted	99.8	0.2
			Proxy-Unencrypted	99.8	0.2
			Naïve Bayes	DR	FPR
			Proxy-Encrypted	96.5	21.6
			Proxy-Unencrypted	78.4	3.5

Tranalyzer	Unbalance	Proxy-Encrypted: 28771 Proxy-Unencrypted: 28088	C4.5	DR	FPR
			Proxy-Encrypted	99.7	0.3
			Proxy-Unencrypted	99.7	0.3
			Naïve Bayes	DR	FPR
			Proxy-Encrypted	97.01	38.7
			Proxy-Unencrypted	61.3	2.9
	Balance	Proxy-Encrypted: 28088 Proxy-Unencrypted: 28088	C4.5	DR	FPR
			Proxy-Encrypted	99.7	0.3
			Proxy-Unencrypted	99.7	0.3
			Naïve Bayes	DR	FPR
			Proxy-Encrypted	97.1	38.7
			Proxy-Unencrypted	61.3	2.9

7.1.6. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote”: 3 Classes Classification

In this case, our aim is to differentiate unencrypted traffic without any proxies from unencrypted local proxy traffic as well as unencrypted remote proxy traffic. Here the words “local” and “remote” are used with respect to the client. Our results show that this is basically a very difficult problem. The false alarm rates are very high when we try to make a prediction about the location (with respect to the client) of the proxy.

Table 15: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote” Classification

Class	Dataset Number
NoProxy-Unencrypted	1
Proxy-Unencrypted-Local	8, 9, 10, 11, 12, 13
Proxy-Unencrypted-Remote	2, 3, 4, 5, 6, 7

Table 16: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted-local: 7332 Proxy-Unencrypted-Remote: 6620	C4.5	DR	FPR
			NoProxy-Unencrypted	72.8	1.9
			Proxy-Unencrypted-Local	69.6	26.2
			Proxy-Unencrypted-Remote	69.7	26.5

Tranalyzer			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	41.9	12.7
			Proxy-Unencrypted-Local	5.6	4.6
			Proxy-Unencrypted-Remote	82.6	78
	Balance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted-local: 1127 Proxy-Unencrypted-Remote: 1127	C4.5	DR	FPR
			NoProxy-Unencrypted	88	7.1
			Proxy-Unencrypted-Local	62.3	16.5
			Proxy-Unencrypted-Remote	65	18.7
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	43	14.7
			Proxy-Unencrypted-Local	5.8	3
			Proxy-Unencrypted-Remote	80.7	67.6
	Unbalance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted-local: 14753 Proxy-Unencrypted-Remote: 13335	C4.5	DR	FPR
			NoProxy-Unencrypted	74.9	1.4
			Proxy-Unencrypted-Local	62.6	32.6
			Proxy-Unencrypted-Remote	63.3	32.3
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	26.6	7.2
			Proxy-Unencrypted-Local	3.9	3.5
			Proxy-Unencrypted-Remote	89.2	86.4
	Balance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted-local: 2323 Proxy-Unencrypted-Remote: 2323	C4.5	DR	FPR
			NoProxy-Unencrypted	88.4	7.4
			Proxy-Unencrypted-Local	51	19.1
			Proxy-Unencrypted-Remote	61.9	22.8
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	23.5	8.6
			Proxy-Unencrypted-Local	70.3	65.3
			Proxy-Unencrypted-Remote	20.6	18.8

7.1.7. “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote”: 3 Classes Classification

In this case, our aim is to differentiate encrypted traffic without any proxies from encrypted local proxy traffic as well as encrypted remote proxy traffic. Again, the words “local” and “remote” are used with respect to the client. Our results show that we can do this better than the previous case. We can achieve over 85% detection with less than 8% false alarm rates.

Table 17: Dataset Numbers for “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote” Classification

Class	Dataset Number
NoProxy-Encrypted	14
Proxy-Encrypted-Local	16
Proxy-Encrypted-Remote	15

Table 18: Results of “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy-Encrypted: 1278 Proxy-Encrypted-Local: 6438 Proxy-Encrypted-Remote: 7946	C4.5	DR	FPR
			NoProxy-Encrypted	93.9	0.4
			Proxy-Encrypted-Local	91.3	6.1
			Proxy-Encrypted-Remote	93.4	7.1
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	12.3	2.3
			Proxy-Encrypted-Local	59.8	48.3
			Proxy-Encrypted-Remote	50.6	36.9
	Balance	NoProxy-Encrypted: 1278 Proxy-Encrypted-Local: 1278 Proxy-Encrypted-Remote: 1278	C4.5	DR	FPR
			NoProxy-Encrypted	95.4	2.2
			Proxy-Encrypted-Local	85.1	8.3
			Proxy-Encrypted-Remote	84.7	7
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	14.9	2.1
			Proxy-Encrypted-Local	73.6	67
			Proxy-Encrypted-Remote	37.1	18.1
Tranalyzer	Unbalance	NoProxy-Encrypted: 2556 Proxy-Encrypted-Local: 12877 Proxy-Encrypted-Remote: 15894	C4.5	DR	FPR
			NoProxy-Encrypted	93.3	0.5
			Proxy-Encrypted-Local	91.7	5.1
			Proxy-Encrypted-Remote	94.3	6.9
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	15.8	3.4
			Proxy-Encrypted-Local	85.2	83.9
			Proxy-Encrypted-Remote	11.4	10.8
	Balance	NoProxy-Encrypted: 2556 Proxy-Encrypted-Local: 2556 Proxy-Encrypted-Remote: 2556	C4.5	DR	FPR
			NoProxy-Encrypted	94.9	2.4
			Proxy-Encrypted-Local	86.9	5.7
			Proxy-Encrypted-Remote	90.3	5.8
			Naïve Bayes	DR	FPR
			NoProxy-Encrypted	16.9	3.6
			Proxy-Encrypted-Local	85.2	80.7
			Proxy-Encrypted-Remote	12	8.7

7.1.8. “NoProxy” vs “Proxy-Local” vs “Proxy-Remote”: 3 Classes Classification

In this case, our aim is to differentiate traffic without any proxies from local proxy traffic as well as remote proxy traffic. In this case, we do not take into account whether they are encrypted or not. Again, as soon as unencrypted proxy and non-proxy traffic is mixed, false alarm rates increase over 10%.

Table 19: Dataset Numbers for “NoProxy” vs “Proxy-Local” vs “Proxy-Remote” Classification

Class	Dataset Number
NoProxy	1, 14
Proxy-Local	8, 9, 10, 11, 12, 13, 16
Proxy-Remote	2, 3, 4, 5, 6, 7, 15

Table 20: Results of “NoProxy” vs “Proxy-Local” vs “Proxy-Remote” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy: 2405 Proxy-Local: 13770 Proxy-Remote: 14566	C4.5	DR	FPR
			NoProxy	80.4	1.4
			Proxy-Local	79.6	16.3
			Proxy-Remote	81.7	17.3
			Naïve Bayes	DR	FPR
			NoProxy	16.2	3.6
			Proxy-Local	5.2	5.0
			Proxy-Remote	91.1	89.5
	Balance	NoProxy: 2405 Proxy-Local: 2405 Proxy-Remote: 2405	C4.5	DR	FPR
			NoProxy	89.8	5.5
			Proxy-Local	72.8	13.3
			Proxy-Remote	73.8	13.1
			Naïve Bayes	DR	FPR
			NoProxy	16.8	4.2
			Proxy-Local	7.5	3.9
			Proxy-Remote	90.9	84.3
Tranalyzer	Unbalance	NoProxy: 4879 Proxy-Local: 27630 Proxy-Remote: 29229	C4.5	DR	FPR
			NoProxy	81.9	1
			Proxy-Local	75.2	17.9
			Proxy-Remote	80	21.2
			Naïve Bayes	DR	FPR
			NoProxy	31.8	5.6

Balance	NoProxy: 4879 Proxy-Local: 4879 Proxy-Remote: 4879	Proxy-Local	5.3	4.2
		Proxy-Remote	90.4	85.1
		C4.5	DR	FPR
		NoProxy	90.6	5.5
		Proxy-Local	69.9	14.9
		Proxy-Remote	70.6	14.1
		Naïve Bayes	DR	FPR
		NoProxy	21.3	6.6
		Proxy-Local	5.5	4.4
		Proxy-Remote	90.3	80.5

7.1.9. “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache”: 3 Classes Classification

In this case, our aim is to differentiate unencrypted traffic without any proxies from cache-proxy unencrypted traffic as well as no-cache-proxy unencrypted traffic. Again, as soon as unencrypted proxy and non-proxy traffic is mixed, it becomes very challenging to predict whether the proxy works with cache or not.

Table 21: Dataset Numbers for “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache” Classification

Class	Dataset Number
NoProxy-Unencrypted	1
Proxy-Unencrypted-Cache	2, 3, 4, 5, 8, 9, 10, 11
Proxy-Unencrypted-NoCache	6, 7, 12, 13

Table 22: Results of “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted-Cache: 9208 Proxy-Unencrypted-NoCache: 4744	C4.5	DR	FPR
			NoProxy-Unencrypted	74.8	1.9
			Proxy-Unencrypted-Cache	86.3	63.8
			Proxy-Unencrypted-NoCache	23.6	11.3
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	69.2	20.6
			Proxy-Unencrypted-Cache	66.3	59.3
			Proxy-Unencrypted-NoCache	13.8	11.4
	Balance	NoProxy-Unencrypted: 1127	C4.5	DR	FPR
			NoProxy-Unencrypted	88.5	7.3

Tranalyzer		Proxy-Unencrypted-Cache: 1127 Proxy-Unencrypted-NoCache: 1127	Proxy-Unencrypted-Cache	41.4	17.7
			Proxy-Unencrypted-NoCache	62.2	28.9
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	36.2	11.3
			Proxy-Unencrypted-Cache	7	7.1
			Proxy-Unencrypted-NoCache	82.8	68.6
	Unbalance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted-Cache: 18536 Proxy-Unencrypted-NoCache: 9552	C4.5	DR	FPR
			NoProxy-Unencrypted	74.7	1.2
			Proxy-Unencrypted-Cache	89.9	67.4
			Proxy-Unencrypted-NoCache	20.3	8.3
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	33.7	10.2
			Proxy-Unencrypted-Cache	61.4	55.5
			Proxy-Unencrypted-NoCache	30.8	28
	Balance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted-Cache: 2323 Proxy-Unencrypted-NoCache: 2323	C4.5	DR	FPR
			NoProxy-Unencrypted	87.1	8
			Proxy-Unencrypted-Cache	50.4	23.2
			Proxy-Unencrypted-NoCache	52.7	23.7
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	46.6	18.3
			Proxy-Unencrypted-Cache	6.7	4.7
			Proxy-Unencrypted-NoCache	77.1	61.9

7.1.10. "NoProxy-Unencrypted" vs "Proxy-Unencrypted-Header" vs "Proxy-Unencrypted-NoHeader": 3 Classes Classification

In this case, our aim is to differentiate the traffic described in the previous case using not only flow information but also combining it whether we observe anything similar to proxy header information. Here we make the assumption that if we find proxy header type information in the traffic we will assume that there is cache-proxy, otherwise we will assume that there is proxy traffic but with no cache. Actually, this assumption improves the performance we have seen in section 7.1.9. Now, the detection rate is above 85% and the false alarm rate is less than 8%.

Table 23: Dataset Numbers for "NoProxy-Unencrypted" vs "Proxy-Unencrypted-Header" vs "Proxy-Unencrypted-NoHeader" Classification

Class	Dataset Number
NoProxy-Unencrypted	1
Proxy-Unencrypted-Header	2, 3, 6, 8, 9, 12
Proxy-Unencrypted-NoHeader	4, 5, 7, 10, 11, 13

Table 24: Results of "NoProxy-Unencrypted" vs "Proxy-Unencrypted-Header" vs "Proxy-Unencrypted-NoHeader" Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted-Header: 6960 Proxy-Unencrypted-NoHeader: 6992	C4.5	DR	FPR
			NoProxy-Unencrypted	73.8	1.3
			Proxy-Unencrypted-Header	98.2	2.2
			Proxy-Unencrypted-NoHeader	96.2	4
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	60.5	19.2
			Proxy-Unencrypted-Header	22.4	13.3
			Proxy-Unencrypted-NoHeader	65	56.1
	Balance	NoProxy-Unencrypted: 1127 Proxy-Unencrypted-Header: 1127 Proxy-Unencrypted-NoHeader: 1127	C4.5	DR	FPR
			NoProxy-Unencrypted	88.2	7.9
			Proxy-Unencrypted-Header	92.1	3.7
			Proxy-Unencrypted-NoHeader	85.5	5.5
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	86.2	32.6
			Proxy-Unencrypted-Header	9.3	3.6
			Proxy-Unencrypted-NoHeader	62	35
Tranalyzer	Unbalance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted-Header: 14009 Proxy-Unencrypted-NoHeader: 14079	C4.5	DR	FPR
			NoProxy-Unencrypted	73.5	1.1
			Proxy-Unencrypted-Header	97.8	3
			Proxy-Unencrypted-NoHeader	95.9	4.4
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	31.3	9.7
			Proxy-Unencrypted-Header	4.6	2.1
			Proxy-Unencrypted-NoHeader	88.2	83
	Balance	NoProxy-Unencrypted: 2323 Proxy-Unencrypted-Header: 2323 Proxy-Unencrypted-NoHeader: 2323	C4.5	DR	FPR
			NoProxy-Unencrypted	86.9	6.5
			Proxy-Unencrypted-Header	92.5	4.4
			Proxy-Unencrypted-NoHeader	86	6.4
			Naïve Bayes	DR	FPR
			NoProxy-Unencrypted	27.6	7.8
			Proxy-Unencrypted-Header	12	8.8
			Proxy-Unencrypted-NoHeader	83	72.1

7.1.11. “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader”: Binary Classification

In this case, our aim is to differentiate proxy unencrypted traffic with header from without header. Our results show over 98% detection with less than 2% false alarm rates.

Table 25: Dataset Numbers for “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” Classification

Class	Dataset Number
Proxy-Unencrypted-Header	2, 3, 6, 8, 9, 12
Proxy-Unencrypted-NoHeader	4, 5, 7, 10, 11, 13

Table 26: Results of “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” Classification

Features	Data set	Number of Instances	Results		
Netmate	Unbalance	Proxy-Unencrypted-Header: 6960 Proxy-Unencrypted-NoHeader: 6992	C4.5	DR	FPR
			Proxy-Unencrypted-Header	98.5	1.8
			Proxy-Unencrypted-NoHeader	98.2	1.5
			Naïve Bayes	DR	FPR
			Proxy-Unencrypted-Header	30.7	21.2
			Proxy-Unencrypted-NoHeader	78.8	69.3
	Balance	Proxy-Unencrypted-Header: 6960 Proxy-Unencrypted-NoHeader: 6960	C4.5	DR	FPR
			Proxy-Unencrypted-Header	98.6	2
			Proxy-Unencrypted-NoHeader	98	1.7
			Naïve Bayes	DR	FPR
			Proxy-Unencrypted-Header	32.1	20.4
			Proxy-Unencrypted-NoHeader	79.6	67.9
Tranalyzer	Unbalance	Proxy-Unencrypted-Header: 14009 Proxy-Unencrypted-NoHeader: 14079	C4.5	DR	FPR
			Proxy-Unencrypted-Header	98.1	2.7
			Proxy-Unencrypted-NoHeader	97.3	1.9
			Naïve Bayes	DR	FPR
			Proxy-Unencrypted-Header	8.1	4.6
			Proxy-Unencrypted-NoHeader	95.4	91.9
	Balance	Proxy-Unencrypted-Header: 14009 Proxy-Unencrypted-NoHeader: 14009	C4.5	DR	FPR
			Proxy-Unencrypted-Header	98.2	2.5
			Proxy-Unencrypted-NoHeader	97.5	1.8
			Naïve Bayes	DR	FPR
			Proxy-Unencrypted-Header	7.8	4.7
			Proxy-Unencrypted-NoHeader	95.3	92.2

7.2. Analysis of results

When we analyzed the results presented in section 7.1, we see that identifying proxy traffic on a server (host) outside of the proxy network is a very challenging problem. The main reason behind this is the fact that proxy behavior is very diverse. The diversity is caused by: (i) the different kinds of proxies used, i.e. HTTP proxy, Cache proxy, SIP proxy, etc., (ii) the location (relative to the client) of the proxy used, and (iii) whether the traffic is encrypted or not.

We have looked into 11 different cases using 16 different traffic files to investigate whether we can differentiate such diverse traffic behaviours using a machine learning approach. Our results show that, our approach is promising when C4.5 machine learning technique is used to classify different behaviours using Netmate traffic flow exporter under balanced data set conditions.

Specifically, we obtain very high performances when we try to differentiate proxy behavior under encrypted traffic conditions. The problem is more challenging when traffic is unencrypted. However, under those conditions if proxy header information is available again our performance is very promising.

Table 27 shows the features employed by the trained models of C4.5 based classifier for each case we evaluated in section 7.1. The different evaluation cases are represented by the different columns of the table from 1 to 11, where:

- 1: Represents “NoProxy-Unencrypted” vs “Proxy-Unencrypted” case
- 2: Represents “NoProxy-Encrypted” vs “Proxy-Encrypted” case
- 3: Represents “NoProxy” vs “Proxy” case
- 4: Represents “NoProxy” vs “Proxy-Encrypted” vs “Proxy-Unencrypted” case

- 5: Represents “Proxy-Encrypted” vs “Proxy-Unencrypted” case
- 6: Represents “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Local” vs “Proxy-Unencrypted-Remote” case
- 7: Represents “NoProxy-Encrypted” vs “Proxy-Encrypted-Local” vs “Proxy-Encrypted-Remote” case
- 8: Represents “NoProxy” vs “Proxy-Local” vs “Proxy-Remote” case
- 9: Represents “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Cache” vs “Proxy-Unencrypted-NoCache” case
- 10: Represents “NoProxy-Unencrypted” vs “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” case
- 11: Represents “Proxy-Unencrypted-Header” vs “Proxy-Unencrypted-NoHeader” case

Table 27: Features employed in the trained models of the classifiers

All Features	1	2	3	4	5	6	7	8	9	10	11
total_fpackets	√	√	√	√	√	√	√	√	√	√	√
total_fvolume	√	√	√	√		√	√	√	√	√	√
total_bpackets	√	√	√	√		√		√	√	√	√
total_bvolume	√	√	√	√		√	√	√	√	√	√
min_fpktl	√	√	√	√	√	√	√	√	√	√	√
mean_fpktl	√	√	√	√	√	√	√	√	√	√	√
max_fpktl	√	√	√	√		√	√	√	√	√	√
std_fpktl	√		√	√	√	√	√	√	√	√	√
min_bpktl	√	√	√	√		√	√	√	√	√	√
mean_bpktl		√	√	√	√	√	√	√	√	√	√
max_bpktl	√	√	√	√	√	√	√	√	√	√	√
std_bpktl	√	√	√	√		√	√	√	√	√	√
min_fiat	√	√	√	√		√	√	√	√	√	√
mean_fiat	√	√		√		√	√	√	√	√	√
max_fiat	√		√	√	√	√	√	√	√	√	√
std_fiat	√		√	√		√	√	√	√	√	√
min_biat	√	√	√	√	√	√	√	√	√	√	√
mean_biat		√	√	√	√	√	√	√	√	√	
max_biat	√		√	√	√	√	√	√	√	√	√
std_biat	√		√	√		√	√	√	√	√	√
duration	√	√	√	√		√	√	√	√	√	√

min_active	√		√	√	√	√	√	√	√	√	√
mean_active	√		√		√	√		√	√		√
max_active						√		√			
std_active				√		√	√	√	√	√	
min_idle	√		√	√		√	√	√	√	√	√
mean_idle	√		√	√	√	√		√	√		
max_idle				√		√		√	√	√	√
std_idle	√		√		√	√		√	√	√	√
sflow_fpackets	√		√	√		√	√	√	√	√	√
sflow_fbytes	√		√	√	√	√		√			
sflow_bpackets				√				√			
sflow_bbytes								√		√	√
fpsh_cnt	√		√	√	√	√	√	√	√	√	√
bpsht_cnt	√	√	√	√	√	√	√	√	√	√	√
furg_cnt						√	√	√	√	√	
burg_cnt											
total_fhlen	√		√	√							
total_bhlen	√		√	√				√	√	√	√
COUNT	30	17	31	33	17	34	27	37	33	32	30

The last row of the table shows how many features out of the 44 features Netmate uses are important for the different evaluation cases (scenarios) studied in this work. These results indicate that Netmate features selected by C4.5 under different cases seems to estimate the delay and size of the flows similar to the passive measurement techniques used on packets but without their limitations.

8. PROPOSED TRAFFIC DE-ANONYMIZER SYSTEM

After the results obtained in section 7, we designed and developed a prototype of our proposed traffic de-anonymizer system based on our C4.5 machine learning based approach to analyze a given traffic file to identify the proxy devices. It should be noted here that our system

could analyze the traffic captured outside of the proxy/firewall network. In other words, it does not require any access to the proxy and/or to the client.

In this section, we show the graphical user interface (GUI) of our prototype system by giving screenshots, Figure 13, from each step that a user goes through while analyzing the high level proxy behavior in a given traffic trace using our system. As discussed earlier, to be able to use this system the captured network traffic traces have to be converted into traffic flows using a tool such as Netmate [14]. Currently, the prototype system only accepts Netmate output files with 44 columns (features) and the *.netmate* extension. However, it can be extended easily to use other flow exporter tools if the expert wants to use a tool different than Netmate.

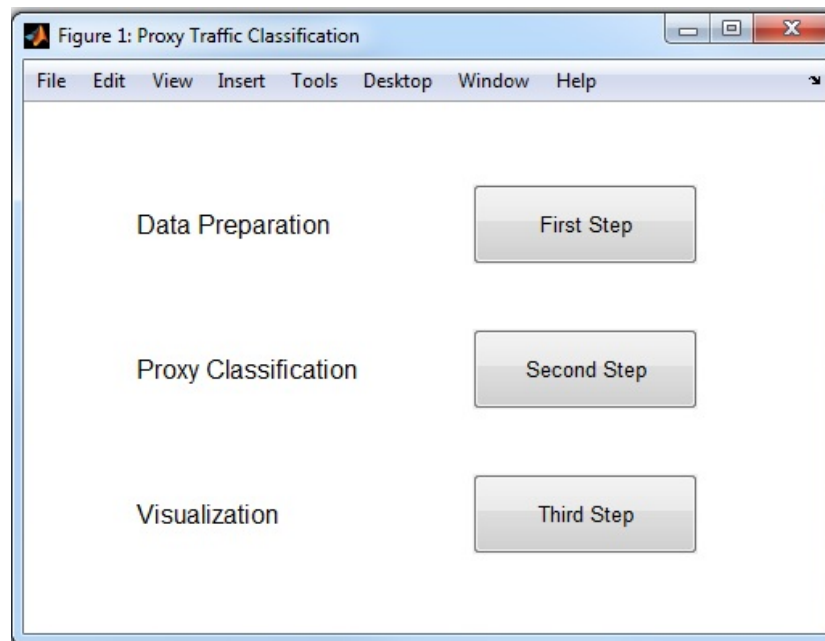


Figure 13: A screenshot of the Prototype GUI

8.1. First Step – Data Preparation:

If a directory is given as input, our system (prototype) will also check subdirectories automatically to find .netmate extension files. Therefore, if a directory with lots of subdirectories is given, it may take some time to search all subdirectories. It is recommended to put Netmate output files in a directory without subdirectories and to give to the system the exact address of this directory. After selecting the correct directory, the system will read all Netmate output files and will apply broken flow test on every flow. The outputs of this step are two files:

(i) 39Columns: This file contains all unbroken flows. For each flow, it has 39 attributes, which were presented in Table 3. The format of this file is arff.

(ii) 50Columns: This file is like the “39Columns” file except that in addition to 39 attributes, it has the source IP address, the source port number, the destination IP address, the destination port number, and the protocol for each flow. The format of this file is “comma delimited” csv.

8.2. Second Step – Proxy Classification:

In our GUI prototype, we have employed three classifiers, which are “Proxy” vs “No-Proxy”, “Encrypted Proxy” vs “Unencrypted Proxy”, and “Unencrypted Proxy with Header” vs “Unencrypted Proxy without Header”, based on the C4.5 models, the Netmate features, and the balanced training datasets.

When the “Second Step” button is clicked, all the input traffic goes through the aforementioned three classifiers and then a folder called “LogFiles” is created in the program executable path. Log files will be separated into “Proxy” and “NoProxy” subdirectories. Each of these subdirectories has other subdirectories, which separate flows based on their behaviors. In each log file, for each flow, the source IP, the source port, the destination IP and the destination

port are stored. It should be noted here that when the program is running, this part might take several minutes to complete. Thus, when you run this step, please let the program to continue. When this step is completed, a popup window will appear.

8.3. Third Step – Analyzing Proxy Behavior:

After clicking on the Third Step button, two files with *text* and *xml* formats will be create in the “data” directory. The *text* file is the input file of the “treemap” open source software, which is employed for visualizing the input data in the “Rectangle View”. On the other hand, the *xml* file is the input file of the “spacetree” open source software, which is employed for visualizing the input data in the “Tree View”. So at this stage, the following window will appear, Figure 14.

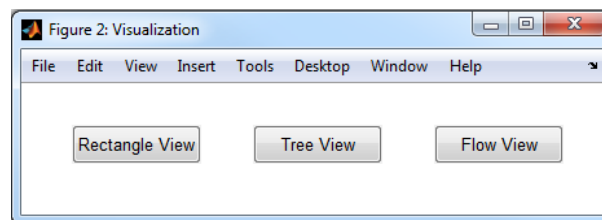


Figure 14: A screenshot of the window for selecting the views

Clicking on the “Rectangle View” button runs “treemap” program and loads the text file as its input. Figure 15 is the rectangle views of the proxy classification of our dataset.

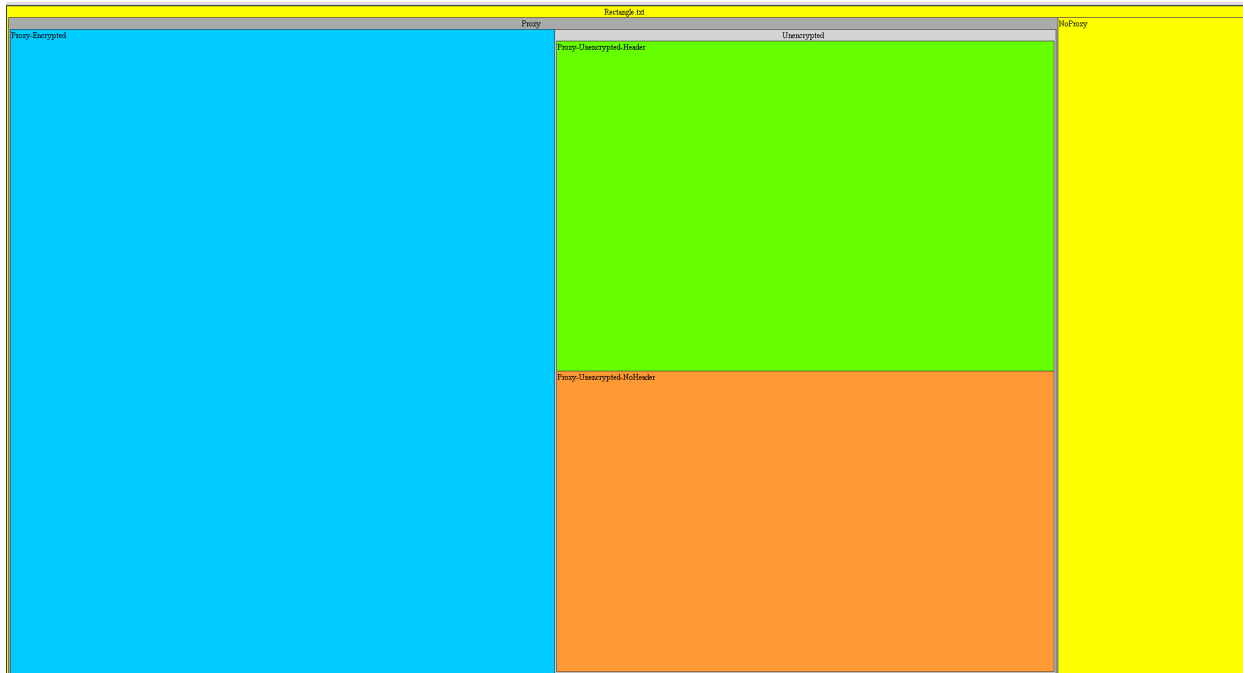


Figure 15: Rectangle view for the Proxy classification of the NIMS Proxy Dataset

In Figure 15, the dimensions of rectangles are based on the numbers of flows of a specific category. As can be seen in these figures, the input data has been classified into No-Proxy (Yellow rectangle) and Proxy (left big rectangle). The Proxy class is also classified into Encrypted (Blue rectangle) and Unencrypted. The Unencrypted Proxy class is then classified into “Unencrypted Proxy with Header” (Green rectangle) and “Unencrypted Proxy without Header” (Orange rectangle).

Clicking on the “Tree View” button runs the “spacetree” program and loads the xml file as its input. Figure 16 shows the classification tree views of the proxy traffic data sets. In this figure, the number in each node shows the number of occurrences of the specific type (class) of flow it represents. As can be seen in this figure, the input data has been classified into Proxy and No-Proxy nodes based on the trained C4.5 classification models. For each of these two nodes, there are other sub-nodes, which are identified based on the other C4.5 trained classification models.

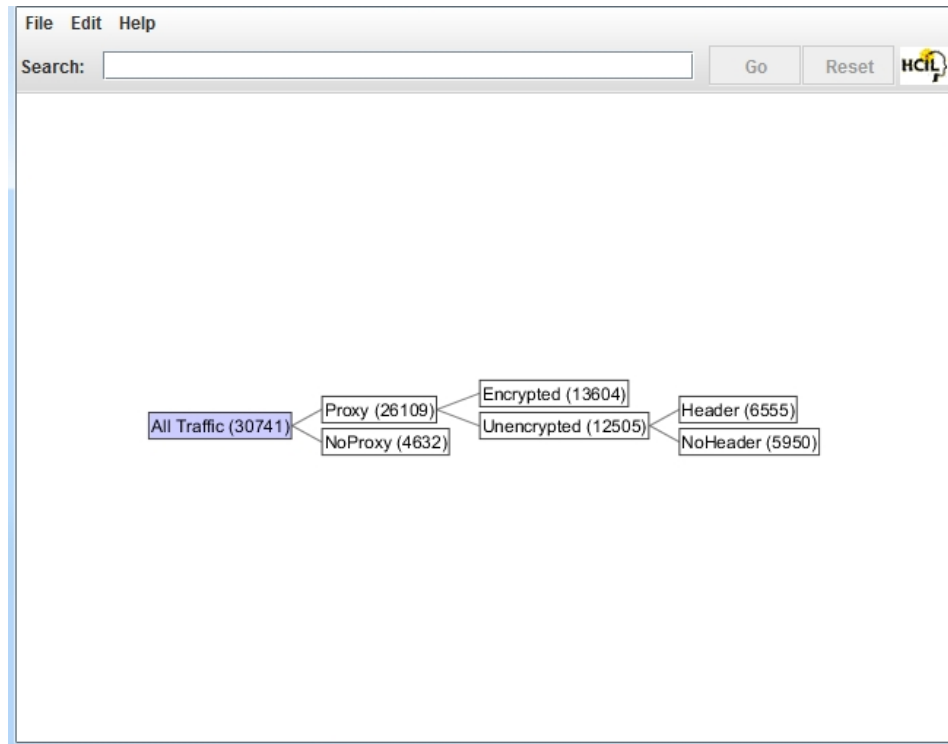


Figure 16: Tree view for classification of the NIMS Proxy Datasets

Clicking on the “Flow View” button, you can see a pop up menu like Figure 17.

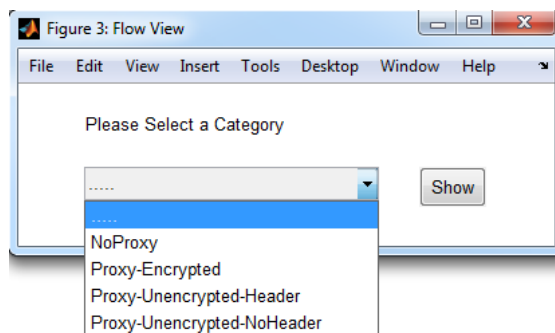
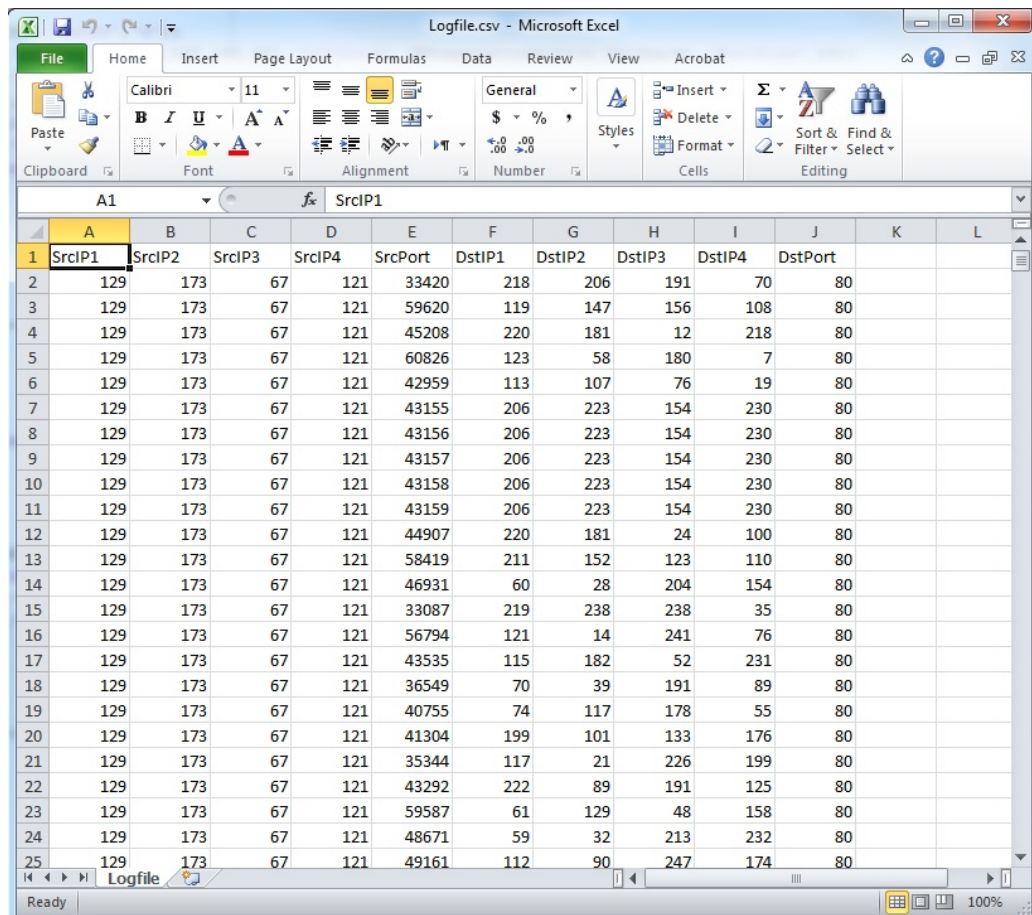


Figure 17: A Screenshot of the Window of selecting the Log File Category

There are four options, NoProxy, Proxy-Encrypted, Proxy-Unencrypted-Header, and Proxy-Unencrypted-NoHeader. By selecting any of the aforementioned options and then clicking on the Show button, the relevant log file will be presented. Figure 18 presents an example of such a log file.



	A	B	C	D	E	F	G	H	I	J	K	L
	SrcIP1	SrcIP2	SrcIP3	SrcIP4	SrcPort	DstIP1	DstIP2	DstIP3	DstIP4	DstPort		
2	129	173	67	121	33420	218	206	191	70	80		
3	129	173	67	121	59620	119	147	156	108	80		
4	129	173	67	121	45208	220	181	12	218	80		
5	129	173	67	121	60826	123	58	180	7	80		
6	129	173	67	121	42959	113	107	76	19	80		
7	129	173	67	121	43155	206	223	154	230	80		
8	129	173	67	121	43156	206	223	154	230	80		
9	129	173	67	121	43157	206	223	154	230	80		
10	129	173	67	121	43158	206	223	154	230	80		
11	129	173	67	121	43159	206	223	154	230	80		
12	129	173	67	121	44907	220	181	24	100	80		
13	129	173	67	121	58419	211	152	123	110	80		
14	129	173	67	121	46931	60	28	204	154	80		
15	129	173	67	121	33087	219	238	238	35	80		
16	129	173	67	121	56794	121	14	241	76	80		
17	129	173	67	121	43535	115	182	52	231	80		
18	129	173	67	121	36549	70	39	191	89	80		
19	129	173	67	121	40755	74	117	178	55	80		
20	129	173	67	121	41304	199	101	133	176	80		
21	129	173	67	121	35344	117	21	226	199	80		
22	129	173	67	121	43292	222	89	191	125	80		
23	129	173	67	121	59587	61	129	48	158	80		
24	129	173	67	121	48671	59	32	213	232	80		
25	129	173	67	121	49161	112	90	247	174	80		

Figure 18: A sample of Proxy Traffic Log File

In this case, the first 4 columns represents the source IP address, the fifth column is the source port number, the following 4 columns represents the destination IP address, and the last column shows the destination port number of the relevant flows, i.e. flows classified as proxy. Using this

information, the network administrator is able to find the Proxy traffic in the offline tcpdump files or may do further actions or apply some policy on ongoing proxy traffic.

As a summary, when C4.5 based classifiers trained solutions are used to evaluate all the data sets employed in this research using the system we developed above, then the following results are obtained, Tables 28 and 29.

Table 28: Confusion matrix for the prototype

		Predicted Labels			
		Proxy-Encrypted	Proxy-Unencrypted-Header	Proxy-Unencrypted-NoHeader	NoProxy
Real Labels	Proxy-Encrypted	13593	1	2	788
	Proxy-Unencrypted-Header	2	6500	19	439
	Proxy-Unencrypted-NoHeader	0	50	5912	1030
	NoProxy	9	4	17	2375

Table 29: Performance of the prototype proxy traffic analysis system

	DR	FPR
Proxy-Encrypted	94.5	0.07
Proxy-Unencrypted-Header	93.4	0.23
Proxy-Unencrypted-NoHeader	84.6	0.16
NoProxy	98.8	8

9. CONCLUSIONS AND FUTURE WORKS

In this research, we perform a study to identify the traffic coming from different computers behind a proxy device whether the traffic is encrypted or unencrypted (clear). To this end, we employed a machine learning based approach using only traffic flow information. To achieve

this, we evaluated two learning techniques, namely C4.5 and Naïve Bayes, using two different flow exporter, namely Netmate and Tranalyzer, system. In doing so, not only we want to compare performances of different learning techniques but also compare two different flow feature sets given that Netmate and Tranalyzer outputs different flow features for the same traffic capture. Our results show that we can identify different behaviours of the computers behind a proxy device using the C4.5 based classifier with Netmate flow features. Moreover, we can perform this analysis without any access to the proxy machine or the clients behind it. Our analysis shows that the most challenging behaviours are hidden in the unencrypted channels and are under no-cache proxy traffic. Future research will investigate different types of proxies and anonymizers such as Tor in both encrypted and unencrypted tunnels using different flow feature sets to compare against the findings in this work.

References:

- 1) B. Li, E. Erdin, M. H. Gunes, G. Bebis, T. Shipley, Review: An Overview of Anonymity Technology Usage, *Computer Communications*, Elsevier, Vol.36 (12), pp. 1-37, 2013.
- 2) Squid [online]. Available: <http://www.squid-cache.org>
- 3) Wireshark [online]. Available: <http://www.wireshark.org>
- 4) R. Beverly, A robust classifier for passive TCP/IP fingerprinting, In Proc. Conference on PAM, Springer LNCS, pp. 1-10, 2004.
- 5) G. Maier, F. Schneider, A. Feldmann, NAT Usage in Residential Broadband Networks, In Proc. Conference on PAM, Springer LNCS, pp. 32-41, 2011.
- 6) H.W. Hsiao, W.C. Fan, Detecting Stepping Stone with Network Traffic Mining Approach, In Proc. Conference on IEEE ICIC, pp. 1176-1179, 2009.
- 7) S. Sulaiman, S. M. Shamsuddin, F. Forkan, A. Abraham, Autonomous Spy: Intelligent Web Proxy Caching Detection Using Neurocomputing and Particle Swarm Optimization, In Proc. Conference on IEEE ISMA, pp. 1-6, 2009.
- 8) H.C. Wu, S.H. Huang, Neural Network Based Detection of Stepping Stone Intrusion, *Expert Systems with Applications: An International Journal*, Elsevier, Vol. 37, pp. 1431-1437, 2010.
- 9) Z. Fang, Z. Sun, A New Method Based on Action Feature to Control and Identify Proxy, In Proc. Conference on IEEE IHMSC, pp. 223-225, 2011.
- 10) R.M. Lin, Y.C. Chou, K.T. Chen, Stepping Stone Detection at the Server Side, In Proc. Conference on IEEE WSCNC, pp. 964-969, 2011.
- 11) J. R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, 1993.
- 12) E. Alpaydin, Introduction to Machine Learning, MIT Press, 2004.
- 13) L. C. Lucien, Maximum likelihood — An Introduction, *ISI Review* **58** (2): 153–171, 1990.
- 14) Netmate FlowCalc [online]. Available: <http://dan.arndt.ca/nims/calculating-flow-statistics-using-netmate/>
- 15) Tranalyzer [online]. Available: <http://tranalyzer.com/>
- 16) RFC 2722 (1999, October), [online]. Available: <http://tools.ietf.org/html/rfc2722>
- 17) WEKA [online]. Available: <http://www.cs.waikato.ac.nz/ml/weka/>